ARMY RESEARCH LABORATORY

Large-Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) Simulations of the Molecular Crystal αRDX

by Lynn B. Munday and Peter W. Chung

ARL-TR-6579 **August 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-6579 **August 2013**

Large-Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) Simulations of the Molecular Crystal αRDX

Lynn B. Munday and Peter W. Chung
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2013	Final	June 2012

4. TITLE AND SUBTITLE
Large-Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) Simulations of the Molecular Crystal αRDX

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Lynn B. Munday and Peter W. Chung

5d. PROJECT NUMBER
MSRM-HSAI

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CIH-C
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6579

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This reference manual provides instructions for determining atomistic material properties important for modeling dislocations in the energetic molecular crystal RDX using the Large-Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) molecular dynamics simulator. Detailed procedures are provided by describing LAMMPS input files and Linux shell scripts used for determining material properties such as elastic constants, minimized crystal structures from various loading conditions and generalized stacking fault energy surfaces of the minimized structures. Detailed descriptions of Matlab scripts are also provided for post-processing the simulation data. These procedures are well suited to molecular crystals where steric interactions across the slip plane can impede dislocation motion. All LAMMPS input and output files, shell scripts, and Matlab files for post-processing are available for download at:

https://arlpartners2.arl.army.mil/hsai/projects/mesoscale/files/LAMMPS_RDX_GSFsurf.tar.gz

15. SUBJECT TERMS

RDX, molecular crystal, dislocation nucleation, Generalized Stacking fault Energy, LAMMPS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lynn B. Munday
Unclassified	Unclassified	Unclassified	UU	70	19b. TELEPHONE NUMBER (*Include area code*) (410) 278-7511

Contents

List of Figures

List of Tables

1. Purpose

This work outlines the procedures used to determine mechanical properties of the energetic molecular crystal RDX from atomistic simulations using the Large-Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) (*1*), a molecular dynamics simulator. The procedures include compiling LAMMPS; creating a LAMMPS atomic configuration and bonding file for the αRDX crystal reported by Choi and Prince (*2*); implementing the Smith and Bharadwaj (*3*) potential to describe RDX atomic interactions; and documenting procedures for running and post-processing a series of molecular dynamics simulations by Munday et al. (*4*–*6*).

Section 0 provides details specific for compiling LAMMPS on a local Linux computer and the U.S. Army Research Laboratory (ARL) Defense Supercomputing Resource Center (DSRC) by using the available makefiles that come with the LAMMPS distribution. Section 5 provides details on implementing the RDX potential energy function given by Smith and Bhardwaj (*3*) using the LAMMPS potential commands. The simulations procedures presented in section 6 were used by Munday et al. (*4, 5*) to determine the material response of αRDX to oriented loads such as the orthotropic elastic constants and phase transitions. These simulations also provided the initial configuration for the stacking fault simulations presented in section 7. The Generalized Stacking Fault (GSF) energy surface from the stacking fault simulations for αRDX were reported by Munday et al. (*4, 6*). Figure 1 presents a process diagram of the steps taken to run an entire stacking fault simulation starting from the experimental αRDX structure. All data files and scripts described in this work are available for download at:

https://arlpartners2.arl.army.mil/hsai/projects/mesoscale/files/LAMMPS_RDX_GSFsurf.tar.gz

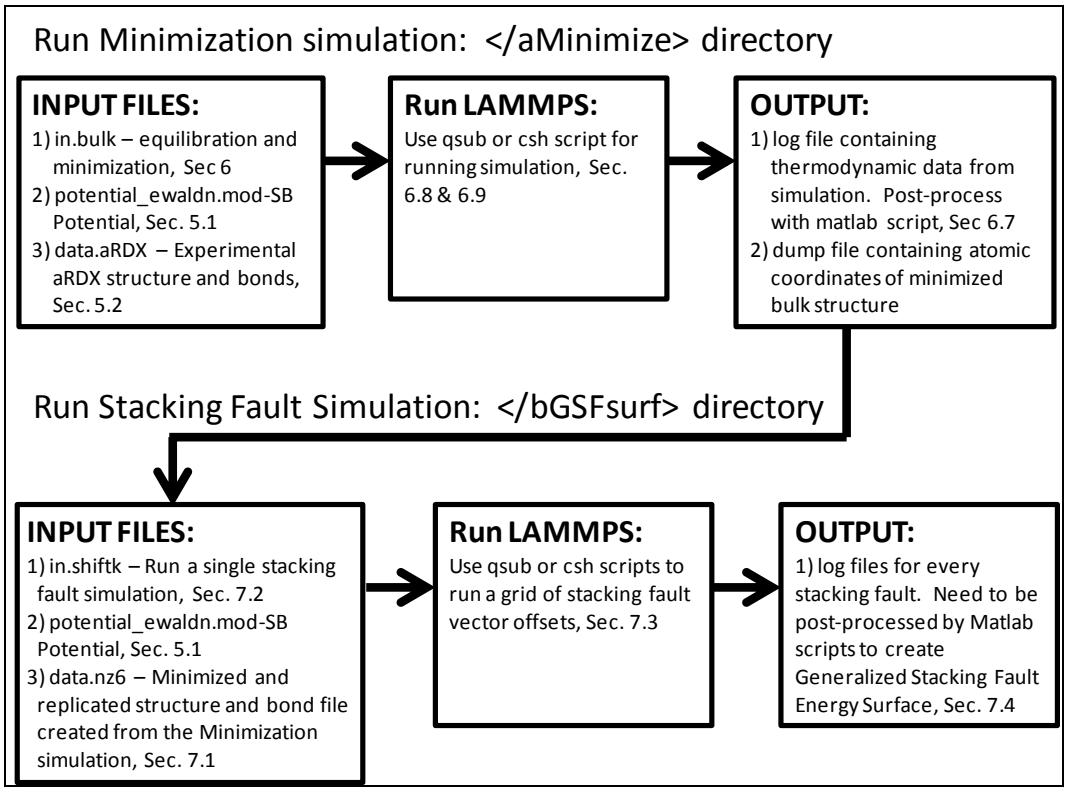

Figure 1. Diagram of the steps used to determine the GSF energy surface for a slip plane starting from equilibration and minimization of the experimental crystal structure. Each step in the flowchart references the section containing the description of the files used.

2. Previous Work

The GSF energy surface can determine the slip systems in a material and the structure of the dislocation core. Vitek (*7*) developed the idea behind the GSF and GSF energy surface, and describes it in the following way.

Cut a perfect bulk crystal along a plane. Displace the two crystal halves relative to one another by a displacement vector **f** that is parallel to the cut plane. The resulting structure is a generalized single layer stacking fault. The new faulted structure contains an excess energy compared to the uncut perfect crystal. This excess energy per unit area of the fault is referred to as the GSF energy, $\Psi(\mathbf{f})$. The procedure is repeated for all **f** spanning a repeat unit cell on the fault plane surface, resulting in the GSF energy surface or Ψ-surface.

Local minima on the Ψ-surface indicate the positions, **f**, of metastable stacking faults with an energy equal to $\gamma_{sf}=\Psi(\mathbf{f})$. Metastable stacking faults are related to partial dislocations with a glide plane being the GSF plane and partial Burgers vector, **b**=**f**, of the local minima. Rice (*8*) found the energy barrier to dislocation nucleation from a crack tip was equal to maxima or saddle

points on the Ψ-surface. This energy point is referred to as the unstable stacking fault energy, γ_{usf}.

Rice introduced an alternative interfacial GSF energy Φ(**u**) given as a function of the interfacial displacement discontinuity along the fault plane, **u**. The energy Φ(**u**) is given by

$$\Phi(\mathbf{u}) = \Psi(\mathbf{f}) - E(\mathbf{f}) \tag{1}$$

where $E(\mathbf{f})$ is the elastic shear strain energy. The crystal lattices above and below the slice plane are shifted as rigid bodies and do not undergo shear deformation. Shear strain only occurs across the crystal lattice being sliced and displaced by **f**. For this case, the interfacial displacement is related to the stacking fault vector by

$$\mathbf{u} = \mathbf{f} - h\boldsymbol{\sigma}_{nt,q}/\mu \tag{2}$$

where μ is the shear modulus, h is the height of the lattice normal to the slice plane, and $\boldsymbol{\sigma}_{nt,q}$ are the shear tractions on the slice plane given by

$$\boldsymbol{\sigma}_{nt} = \mathbf{t}\cdot\boldsymbol{\sigma}\cdot\mathbf{n} \text{ and } \boldsymbol{\sigma}_{nq} = \mathbf{q}\cdot\boldsymbol{\sigma}\cdot\mathbf{n} \tag{3}$$

where **σ** is the stress tensor and **n**,**t**,**q** define an orthornormal coordinate system with **n** normal to the slice plane and **t**,**q** are in the slice plane. Then by assuming the material is linear elastic, the shear strain energy is given by

$$E(\mathbf{f}) = h(\boldsymbol{\sigma}:\boldsymbol{\varepsilon})/2 = h(\boldsymbol{\sigma}:\boldsymbol{\sigma})/2\mu = h(|\boldsymbol{\sigma}_{nt}|^2+|\boldsymbol{\sigma}_{nq}|^2)/2\mu \tag{4}$$

where **ε** is the strain tensor and the lattice height, h, is used to convert the elastic strain energy density to strain energy per unit area of fault. The Φ-surface is used to determine dislocation core properties using the Pierls Nabarro model giving the shear stress required to move a dislocation and other dislocation mobility properties.

However, the shear strain is negligible at critical points on the Φ- and Ψ-surfaces and either surface will provide the same values for γ_{usf} or γ_{sf} and their corresponding displacement vectors, **f** and **u**.

3. This Work

The GSF concepts introduced in the previous section work well for atomic and metallic crystals like copper because the slice plane is flat and the energy of a general stacking fault can be sufficiently relaxed by allowing atomic motion normal to the slice plane. Relaxation normal to the slip plane does not allow shear deformation to develop in the crystal halves. Molecular crystals on the other hand contain entire molecules or groups of molecules at each lattice site. The molecules cause the slice planes to be jagged, and in order to avoid molecule overlap, the GSF displacement, **f**, must contain a normal component. The molecules of a GSF can only be

sufficiently relaxed by allowing them to undergo conformation and orientation changes. Ideally, the conformation and orientation changes could be done in a way that would avoid shear deformation of the entire lattice, but this type of constraint is difficult to implement. Therefore, the molecular relaxations implemented allow for motion tangential to the slice plane, resulting in shear deformation of the crystal halves. The additional strain energy, $E(\mathbf{f})$, from the shear deformation is then accounted for in the calculation of $\mathbf{u}$ and $\Phi(\mathbf{u})$.

The next sections outline the procedures used to produce GSF energy surfaces for the energetic molecular crystal RDX using the molecular dynamics software LAMMPS (version lammps-21Dec11). These steps include equilibration and minimization of the initial bulk crystal, creation of a single GSF, the relaxation procedure of the GSF structure, and post-processing to produce the Φ- and Ψ-surfaces.

4. Compiling LAMMPS

The LAMMPS package from Sandia National Laboratories is available in serial and parallel versions that are easily compiled on Linux computers using the makefiles included in the LAMMPS distribution. There is also a precompiled executable for Windows computers with limited capabilities. This work was done using a parallel compilation of the code using the makefiles available in the LAMMPS /src directory. The simulations in this work used a parallel version of the code compiled on the ARL DSRC machines Harold and MJM and also on a local Dell T7500 x86_64 GNU/Linux workstation running CentOS 5.8. Parallel versions require an Message Passing Interface (MPI) library and C++ compilers. These can be loaded automatically as modules on ARL computers. On the local computer, these are available using the /usr/cta/CSE development environment. On the ARL DSRC computers, the modules are automatically available.

The LAMMPS Web site (http://lammps.sandia.gov/) contains directions for building LAMMPS on various architectures. These directions are streamlined below for building the version of LAMMPS used in this work on the Dell T7500 Linux workstation.

1. Edit the file `/src/MAKE/Makefile.openmpi` by deleting the settings for the flags LMP_INC, FFT_INC, and FFT_LIB. This work does not use the particle-particle particle-mesh (PPPM) Ewald sums to calculate electrostatic interactions so there is not a need for fast Fourier transforms (FFTs).

2. Before the code can be compiled, the MPI library must be added to the current path. This is done using the computational science environment (CSE) environment modules by loading the MPI module "`module load cse/openmpi/1.4.1`" or load the latest MPI version. This is the MPI library that is used by "`/src/MAKE/Makefile.openmpi`"

3. In the "/src" directory, type "make yes-USER-EWALDN" to include an Ewald method capable of working with triclinic simulation cells. This is not a standard package. This Ewald method also has the ability to include long range pair interactions in the Ewald sum, but this work does not use this capability.

4. In the "/src" directory type "make package-status" to see that the packages KSPACE, MANYBODY, MOLECULE, and USER-EWALDN are switched to "YES"; they will be built.

5. In the "/src" directory type "make openmpi" to make the executable lmp_openmpi. The executable will be created in the /src directory.

6. To run a simulation use step 2 to load the openmpi 1.4.1 module and type "mpirun -np 4 lmp_openmpi < file.input". This will run a parallel simulation of the lmp_openmpi executable on four processors (-np 4) using the input file file.input. This requires lmp_openmpi and file.input to be located in the same directory and all output will be written to this directory. Detailed execution instructions for running LAMMPS simulations are available on the LAMMPS Web site and sample input files are available in the folder /examples.

On the ARL DSRC Harold cluster, LAMMPS is compiled using the SGI MPI libraries and the Intel compilers, all of which can be loaded by modules. A majority of this work involves a 50-ps equilibration of bulk RDX using periodic simulations cells containing a 3x3x3 unit cell block of RDX (4536 atoms) and 1-fs timesteps. For this small domain size and atom count, all parallel versions of the code take about ~1 h on eight processors, regardless of the MPI version or compiler used.

5. Atomic Configuration and Potential Energy Function

A LAMMPS simulation, or any type of atomistic simulation, requires three sets of data:

1. Atomic configuration data and bond connectivity.

2. Potential energy data describing how the atoms interact.

3. Main input file containing simulation procedures that initialize the simulation by reading in atomic configuration and potential data and direct LAMMPS to run the simulation under specified boundary conditions by using time integrators (ensembles).

In this section, the atomic configuration data and potential are described and their implementation into LAMMPS is given. The structure of the main input file and examples are given later in section 6.

A bond representation of the RDX molecule is shown in figure 2a, where the bonds are colored according the labeled atoms. The bond numbering shown is used in the LAMMPS configuration file to define atom bond connectivity. The room temperature, atmospheric pressure stable crystal structure of RDX is referred to as αRDX. In this work, the atomic coordinates given by Choi and Prince (*2*) are used. Choi and Prince (*2*) used x-ray diffraction to determine the crystal structure of αRDX and found the unit cell to contain eight molecules and belong to the orthorhombic Pbca space group with lattice constants (*a*,*b*,*c*) = (13.182,11.574, 10.709) Å. The Pbca spacegroup symmetry operators are shown in figure 2b with the RDX molecules overlaid on it in the αRDX configuration. Projections of the αRDX unit cell along the different lattice vectors are shown in figure 2c–e. The unit cell can be replicated along the lattice vectors to create a larger crystal.

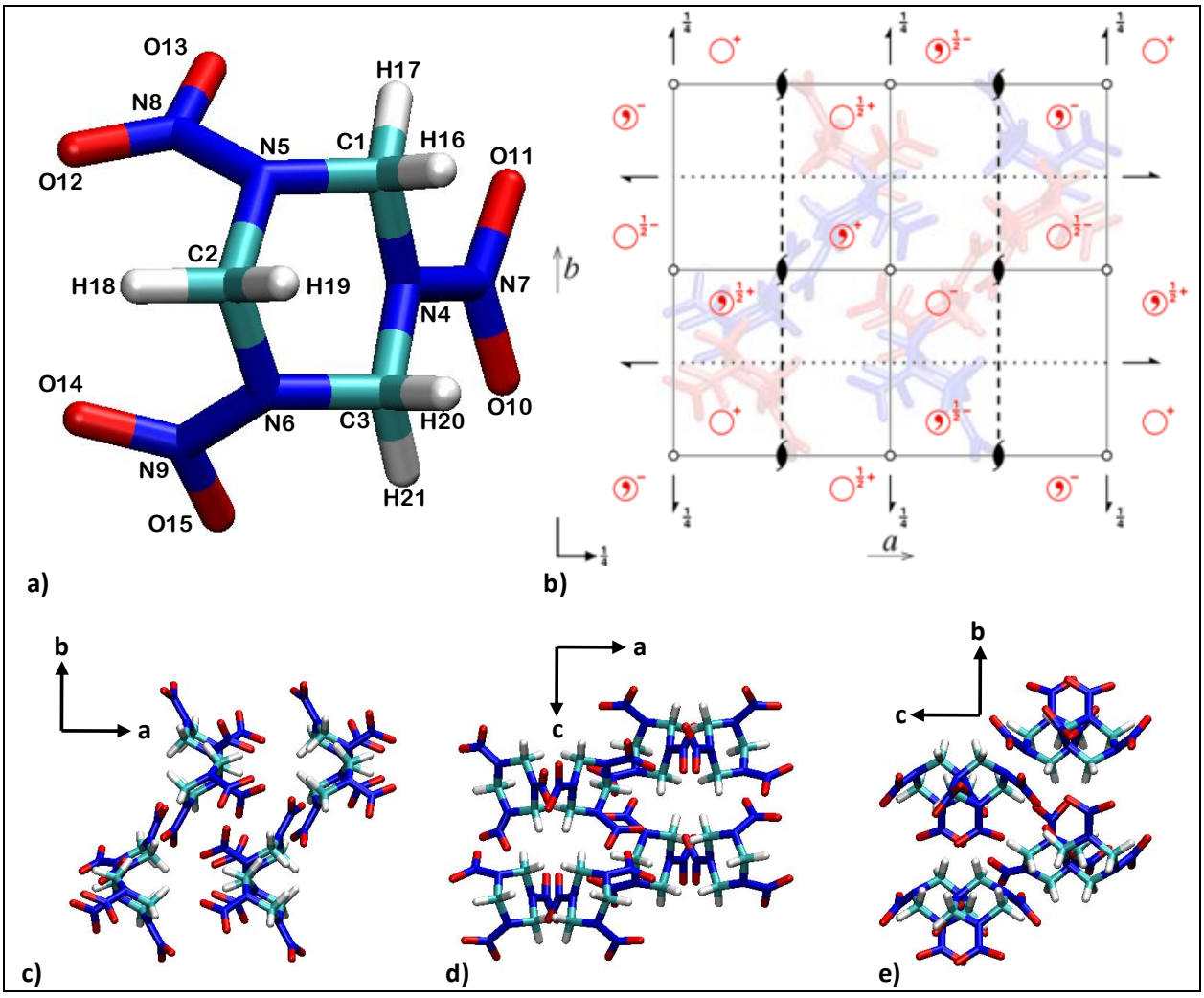

Figure 2. (a) Single RDX molecule and atom numbering. (b) Pbca space group symmetry operators overlaid on RDX molecules arranged in the αRDX crystal structure. Due to inversion symmetry, Red molecules are right handed and blue molecular are left handed. Projections of the αRDX crystal onto the (c) c-axis, (d) b-axis, and (e) a-axis.

The Smith and Bharadwaj (SB) potential (*3*) for HMX is used to describe the nonbonded and bonded interactions of RDX. The form of the potential is given as

$$
\begin{aligned}
U = & \sum_{bonds} \frac{1}{2} K_{ij}^{S} \left(r_{ij} - r_{ij}^{0}\right)^2 + \sum_{angles} \frac{1}{2} K_{ijk}^{B} \left(\theta_{ijk} - \theta_{ijk}^{0}\right)^2 \\
& + \sum_{proper\ dihedrals} \frac{1}{2} K_{ijkl}^{T} \left[1 - cos\left(n\varphi_{ijkl}\right)\right] \\
& + \sum_{improper\ dihedrals} \frac{1}{2} K_{ijkl}^{D} \varphi_{ijkl}^{2} \\
& + \sum_{i=1}^{N-1} \sum_{j>i}^{N} \left(+ A_{ij} exp\left(-B_{ij} r_{ij}\right) - \frac{C_{ij}}{r_{ij}^{6}} + k_e \frac{q_i q_j}{r_{ij}} \right)
\end{aligned}
\tag{5}
$$

The SB potential can be used to describe the flexible bonds of the RDX molecule and nonbonded electrostatic and dispersion repulsion forces. The bonded terms include harmonic bonds, angles, improper dihedrals, and cosine series dihedral interactions. The nonbonded terms include exponential repulsion, r^{-6} dispersion and electrostatic interactions. Constants for the SB potential are given in table 1.

Table 1. SB potential for HMX/RDX (*3*, *9*).

Bond stretches, $U = 0.5K_{ij}^{S}\left(r_{ij} - r_{ij}^{0}\right)^{2}$			
Bond	K_{ij}^{s} (kcal/mol/Å^2)	r_{ij}^{0} (Å)	
O-N	1990.1	1.23	
N-N	991.7	1.36	
N-C	672.1	1.44	
C-H	641.6	1.09	
Valence bends, $U = 0.5K_{ijk}^{B}\left(\theta_{ijk} - \theta_{ijk}^{0}\right)^{2}$			
Angle	K_{ijk}^{B} (kcal/mol/rad^2)	θ_{ijk}^{0} (rad)	
O-N-O	125.0	2.1104	
O-N-N	125.0	1.8754	
N-N-C	130.0	1.6723	
C-N-C	70.0	1.843	
N-C-H	86.4	1.8676	
H-C-H	77.0	1.8938	
N-C-N	70.0	1.9289	
Torsions, $U = 0.5K_{ijkl}^{T}\left[1 - \cos\left(n\phi_{ijkl}\right)\right]$			
Dihedral	K_{ijkl}^{T} (kcal/mol)	n	
O-N-N-C	8.45	2	
O-N-N-C	0.79	4	
O-N-N-C	0.004	8	
H-C-N-C	–0.16	3	
C-N-C-N	3.30	1	
C-N-C-N	–1.61	2	
C-N-C-N	0.11	3	
Out of plane bends $U = 0.5K_{ijk}^{D}\phi_{jikl}^{2}$			
Improper Dihedral	K_{ijk}^{D} (kcal/mol/rad^2)		
C-N-C…*N	8.0	Where …*N is the atom kept in-plane	
O-N-O…*N	89.3		
van-der-Waals interactions, $U = A_{ij}\exp\left(-B_{ij}r_{ij}\right) - C_{ij}/r_{ij}^{6}$			
Atoms pair type	A_{ij} (kcal/mol)	B_{ij} (Å^{-1})	C_{ij} (kcal/mol Å^6)
C···C	14976.0	3.090	640.8
C···H	4320.0	3.415	138.2
C···N	30183.57	3.435	566.03
C···O	33702.4	3.576	505.6
H···H	2649.7	3.740	27.4
H···N	12695.88	3.760	116.96
H···O	14175.97	3.901	104.46
N···N	60833.9	3.780	500.0
N···O	67925.95	3.921	446.6
O···O	75844.8	4.063	398.9
Atomic partial charges			
Atom type	Q		
C	–0.540000		
N(amine)	0.056375		
N(nitro)	0.860625		
O	–0.458500		
H	0.270000		

5.1 LAMMPS Implementation of the SB Potential (*3*)

FILE: potential_ewaldn

The LAMMPS implementation of the SB potential (*3*) given by equation 5 with parameters given in table 1 is presented in this section. The potential file is called from the main LAMMPS input file using the command "`read_data`". This potential file remains unchanged for all of the following simulations presented in this work. This potential file uses the atomic numbering and connectivity used in the LAMMPS atomic configuration file given in section 5.2 and shown in figure 2a. The `ewald/n` kspace solver is used to compute long-range electrostatic interactions with precision given by 1.0e–4. This parameter may need to be changed in newer releases of the LAMMPS code. The ewald/n kspace solver is also a nonstandard package that must be included at compile time (see section 4). A tail correction is also used for long-range van der Waals (VdW) interactions, which may not be wanted for simulations with vacuum layers like those used in for GSFs. The `special_bonds` type is used to allow 1–4 electrostatics/VdW interactions to be included in a dihedral bond as required by the SB potential. Ring and nitro group nitrogens (type #2 and type #3) have a different partial charge in the atomic configuration file and each must be given a `pair_coeff` even though they are the same. All of the cross `pair_coeff` given by the SB potential are found using geometric mixing rules except for the C-H (1–5) interaction where geometric mixing would give 132.5. If `ewald/n` is used to calculate the long-range VdW dispersion energy then geometric mixing is assumed for the C-H (1–5) interaction and it will be assumed that it is 132.5. The other potential terms are straightforward and are common in COMPASS style potentials (*10*). Red lettering and # symbolize comments.

```
bond_style      harmonic # quadratic
angle_style     harmonic # quadratic
dihedral_style     harmonic # cosine series
improper_style     harmonic # improper dihedral quadratic
pair_style      buck/coul/long 10
pair_modify        tail yes # tail correction to vdw cutoff
kspace_style    ewald/n 1.0e-4  #only non-ortho ewald sum

#SB Pot 0.5K in Bedrov et al. (2001) J Comp Matl design paper
pair_coeff  1 1 14976.00 0.323625 640.80
pair_coeff  1 2 30183.57 0.291121 566.03
pair_coeff  1 3 30183.57 0.291121 566.03
pair_coeff  1 4 33702.40 0.279642 505.60
pair_coeff  1 5 4320.000 0.292826 138.2
pair_coeff  2 2 60833.90 0.264550 500.00
pair_coeff  2 3 60833.90 0.264550 500.00
pair_coeff  2 4 67925.95 0.255037 446.60
pair_coeff  2 5 12695.88 0.265957 116.96
pair_coeff  3 3 60833.90 0.264550 500.00
pair_coeff  3 4 67925.95 0.255037 446.60
pair_coeff  3 5 12695.88 0.265957 116.96
pair_coeff  4 4 75844.80 0.246124 398.90
pair_coeff  4 5 14175.97 0.256345 104.46
```

```
pair_coeff  5 5 2649.700 0.267380 27.400

bond_coeff  1 995.05 1.23
bond_coeff  2 495.85 1.36
bond_coeff  3 336.05 1.44
bond_coeff  4 320.80 1.09

angle_coeff 1 62.5 120.917
angle_coeff 2 62.5 107.453
angle_coeff 3 65.0 95.816
angle_coeff 4 35.0 105.596
angle_coeff 5 43.2 107.006
angle_coeff 6 38.5 108.507
angle_coeff 7 35.0 110.518

dihedral_coeff    1  4.225 -1 2
dihedral_coeff    2  0.395 -1 4
dihedral_coeff  3  0.002 -1 8
dihedral_coeff  4 -0.080 -1 3
dihedral_coeff  5  1.650 -1 1
dihedral_coeff  6 -0.805 -1 2
dihedral_coeff  7  0.055 -1 3

improper_coeff    1 4.000 0.0
improper_coeff    2 44.65 0.0

special_bonds     lj/coul 0.0 0.0 1.0 #allowing all topo 1-4 coul/lj
```

5.2 LAMMPS Atomic Configuration and Bond Connectivity

FILE: data.aRDX*

This LAMMPS file contains the simulation cell size, atom types and positions, and bond connectivity and is called from the main LAMMPS input file using the command "`read_data`". The example given below is for a single RDX molecule using the numbering shown in figure 2a. The bonds and their connectivity are given according to the potentials used in the LAMMPS potential file given in section 5.1. There are two types of nitrogens with different partial charges. Type #2 are part of the amine ring and type #3 are part of the nitro group. Every bond in the system must be explicitly defined in the data file. Files containing more molecules, i.e., a unit cell of αRDX containing eight RDX molecules, will be the same format with the bond connectivity section being eight times longer. The Matlab file lammpstrj2data.m is used to create this file from a LAMMPS " `dump custom` " file. Red lettering and # symbolize comments.

```
    21 atoms
    21 bonds
    36 angles
    66 dihedrals
    6 impropers

  5 atom types
```

```
  4 bond types
  7 angle types
  7 dihedral types
  2 improper types

  0.00000 13.18200 xlo xhi
  0.00000 11.57400 ylo yhi
  0.00000 10.70900 zlo zhi
  0.00000  0.00000  0.00000 xy xz yz

Masses
1 12.011000
2 14.007200 #ring or amine nitrogens
3 14.007200 #nitro group nitrogens
4 15.999430
5 1.0080000

Atoms
1    1    1  -0.540000  2.42420000  4.14120000  4.71200000
2    1    1  -0.540000  0.66310000  2.82410000  3.63570000
3    1    1  -0.540000  1.96020000  4.41320000  2.31210000
4    1    2  0.056375  2.32140000  5.04630000  3.56610000
5    1    2  0.056375  1.15610000  3.46760000  4.85870000
6    1    2  0.056375  0.70660000  3.72910000  2.49410000
7    1    3  0.860625  2.97910000  6.22570000  3.58320000
8    1    3  0.860625  0.20430000  4.07980000  5.66930000
9    1    3  0.860625  -0.43900000  4.48490000  2.22530000
10   1    4  -0.458500  2.99230000  6.86570000  2.55730000
11   1    4  -0.458500  3.49190000  6.58210000  4.64560000
12   1    4  -0.458500  -0.91350000  3.63660000  5.63510000
13   1    4  -0.458500  0.59850000  4.94330000  6.41470000
14   1    4  -0.458500  -1.47770000  4.09030000  2.68150000
15   1    4  -0.458500  -0.31110000  5.42940000  1.48860000
16   1    5  0.270000  3.16500000  3.40740000  4.52560000
17   1    5  0.270000  2.65350000  4.70830000  5.61580000
18   1    5  0.270000  -0.34410000  2.45020000  3.79100000
19   1    5  0.270000  1.33800000  2.00000000  3.41830000
20   1    5  0.270000  2.70490000  3.66430000  2.05290000
21   1    5  0.270000  1.89820000  5.12960000  1.51320000

Bonds
     1  1   12    8
... 1-2 bonds 1 to 21
    21  4   21    3

Angles
     1  1   12    8   13
...1-3 bonded angles 1 to 36
    36  7    4    1    5

Dihedrals
     1  1   12    8    5    1
... 1-4 bonded dihedrals 1 to 66
    66  7    2    5    1    4

Impropers
     1  1    4    3    1    7
```

```
... 1-4 bonded improper dihedrals 1 to 6
     6   2     9    14    15     6
```

6. LAMMPS Equilibration and Minimization Input Files

The input files that provide LAMMPS with the commands needed to execute a simulation are presented in this section. The input files are read into LAMMPS sequentially so the order of the commands is important. The following sections present portions of a single input file starting with initialization, equilibration of the structure, and finally, quenching and minimization. The first section of the input file containing the initialization commands is presented in section 6.1. During initialization the simulation is set up, the potential and atomistic configuration and bonding data are read in and output commands for thermodynamic and atomic configuration are given. After the simulation is set up, the simulation is run using the different ensembles as presented in sections 6.2–6.4. After the structure has been equilibrated, sections 6.5 and 6.6 present the commands used to quench and minimize the structure for use as the initial structure for stacking fault and decohesion simulations presented in section 7. The following input files used for equilibration in sections 6.2–6.4 were used by Munday et al. (*4, 5*) to determine the response of bulk αRDX to axial loads, i.e., *elastic constants, phase transitions*.

6.1 Initialization

FILE: in.*

The first set of LAMMPS commands initializes the simulation. In this work, the variable "`s_zz`" is passed into the simulation from the command line and is printed to the output log file containing details of the simulation as the commands are executed. The variable "`s_zz`" is used to set the stress state or pressure in the following simulation examples. Passing in variables like "`s_zz`" from the command line makes the input script applicable to several stress states without having to directly modify the input file. This works well when submitting jobs to the high-performance computer (HPC) using PBS scripts or when writing your own shell scripts.

Several initializing commands are then given to describe the simulation units, dimensionality, periodicity, and style of atoms/bonds in the simulation. The type of "`atom_style` defined must fit the format of the "`read_data`" configuration file and potential files described in section 5. In this example, it is assumed that the file "`data.aRDX`" read in by "`read_data`" only contains data for a single αRDX unit cell and the "`replicate`" command is used to create a simulation cell containing a 3x3x3 unit cell block. The dimensions of the simulation cell must be at least twice as large as the 10-Å cut-off used to calculate nonbonded interactions from the LAMMPS command "`pair_style buck/coul/long 10`" specified in the potential file presented in section 5.1. This input file is continued with three alternative loading scenarios

given in sections 6.2–6.4 to bring the bulk crystal to the desired equilibrium state. In the file below, Red lettering and # symbolize comments.

```
### MINIMIZE RDX BULK at specified stress_zz from pbs
#the stress variable s_zz is passed to the simulation during execution
print "set on command line s_zz=-${s_zz}"   #z-force on rigid body

dimension   3
newton      on   # always on
boundary    p p p # periodic in xyz
units       real  # kcal/mol, fs, atm
atom_style  full  # coulomb and intra bonds
read_data   data.aRDX # reading data file for single unit cell
replicate  3 3 3 # replicate unit cell into 3x3x3 block

include       potential_ewaldn.mod # include potential file from above
thermo     10  # number of steps per output
thermo_style  custom step atoms temp &
              vol lx ly lz xy xz yz & # & lets command span multiple lines
        press pxx pyy pzz pxy pxz pyz pe ke enthalpy &
              evdwl ecoul epair emol elong # data to output

thermo_modify  norm no # do not normalize thermo data by number of atoms

# commands to output atomic configuration files on certain timesteps
dump          1 all custom 10000 all.lammpstrj id type xu yu zu
dump_modify   1 first yes # output on first timestep

# create neighbor list of atoms by binning all atoms within 1.0+pair cuttoff
neighbor    1.0 bin
# build a new pair list every timestep with no delay but only if an atom has
# moved by half the skin distance
neigh_modify      every 1 delay 0 check yes

timestep    1.0 # integration timestep size in fs
#give atoms random velocity with temperature of 300K
velocity    all create 300.0 2349851 dist Gaussian
```

6.2 Uniaxial Stress

FILE: in.bulk

After the simulation cell has been set up during initialization, the atomic trajectories are computed by choosing the proper integration algorithms called ensembles. Different ensembles conserve different system energies that depend on the boundary conditions of the simulation. In this section an NPT ensemble (constant number of particles “N”, controlled pressure or stress “P”, controlled temperature “T”) is used to apply a uniaxial state of stress to a bulk crystal at T = 300 K. The temperature and pressure loads need to be applied gradually during the initial stages of the simulation in order to slowly bring the crystal to the correct temperature and stress state without artificially pushing the system into a higher energy configuration. After the system is brought to the correct state, equilibrated thermodynamic data can be collected and averaged to

determine properties of the system. Simulation results showing this process are presented in figure 3, where the vertical black lines indicate changes to the ensemble used to bring the system to the correct temperature and uniaxial stress state.

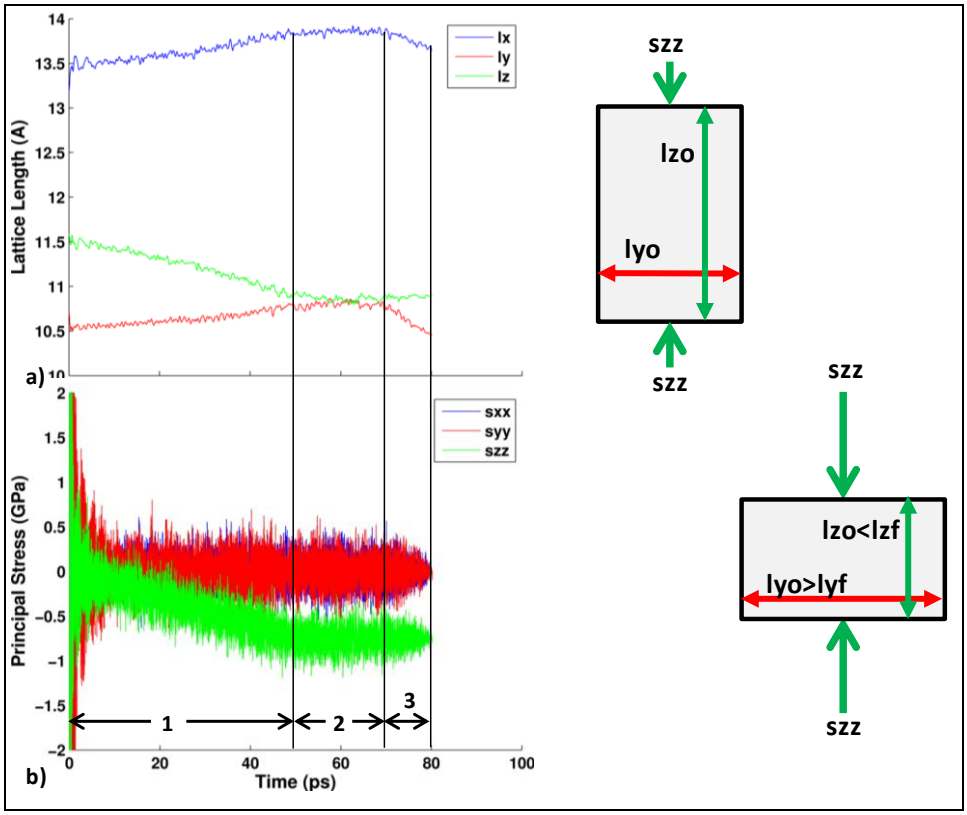

Figure 3. Uniaxial stress results using file in.bulk. (a) Unit cell lattice constants (Å) and (b) principal stresses (GPa) vs. simulation time. The pictures indicate the applied loads and deformation. Regions on plots are labeled by the steps 1–3 given in the input.bulk file.

In figure 3, the following steps given in the input.bulk file are illustrated:

- Step 1: For 50 ps, apply ramped up uniaxial stress from 0 to `${szz}` = –0.5 GPa in the z-direction. From the lattice dimensions given in (a), the z-direction is shown to be along the [010]-lattice dimension.
- Step 2: Equilibrate at fixed uniaxial stress (`${szz}` = –0.5 GPa) for 20 ps.
- Step 3: Langevin quenching using Nsh ensemble (constant orthotropic stress, constant energy), presented in section 6.5. Quenching under the Nsh ensemble allows the lattice to thermally contract.

The LAMMPS commands given next are a continuation of an entire LAMMPS input file and follow the initialization commands presented in section 6.1. The thermodynamic data collected from this input script are shown in figure 3. The input file brings the structure into equilibrium at the specified stress and temperature state through two steps, shown by the black lines in figure 3.

Step 1 ramps up the uniaxial stress applied to the z-direction of the simulation cell from 0 to "`s_zz`" over 50 ps at T=300 K using the "`fix npt`" command. Step 2 changes the "`fix npt`" command to equilibrate the structure at a constant uniaxial stress set to "`s_zz`" for 20 ps. This simulation was set up with the **c**-axis of the unit cell oriented in the z-direction and figure 3a shows the z-axis getting smaller in the region labeled "1" as the uniaxially compressive stress is linearly increased from 0 to s_zz, shown in figure 3b. The other lattice lengths increase due to the Poisson effect while the average stress in these directions stays constant at the set point of 0 GPa. During step 2, the desired uniaxial stress level is reached and the structure is equilibrated, shown by the constant stresses and lattice lengths in the region labeled "2". The elastic constants are computed from the average values obtained during the equilibration step (*4*, *5*). Figure 3 also shows the quenching step by the region labeled by "3", where the temperature is brought from T = 300 K to T = 0 K under zero stress. The decreasing temperature causes the oscillations in the stress field to decrease and the lattice constants also contract to thermal expansion. The input file for quenching and minimization are given in sections 6.5 and 6.6. This input file can be easily modified to apply isotropic pressure or to uniaxially stress the x or y lattice directions. Applied shear stress requires the stress to be applied to the tilt dimensions of the box (xy, xz, yz). See the LAMMPS manual for information about the "`fix NPT`" ensembles. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
#### NPT THERMALIZE AND COMPRESS
#convert stress (GPa) to atmospheres
variable tmp equal ${s_zz}*9869.23267

#Step 1: Apply integration fix to ramp up uniaxial stress in the z direction
# from 0 to s_zz.
fix         1 all npt temp 300 300 100 &
        x 0 0 1000 y 0 0 1000 &
        z 0 ${tmp} 1000 couple none nreset 1000
run         50000 #run npt ensemble for 50000 steps

#Only dump one atomic configuration during equilibration portion of run
dump_modify 1 every 1000000 first yes
# Step 2: Equilibrating at constant uniaxial stress in the z-direction
fix         1 all npt temp 300 300 100 &
        x 0 0 1000 y 0 0 1000 &
        z ${tmp} ${tmp} 1000 couple none nreset 1000
run         20000
unfix       1

write_restart restart.NPT #create a restart file of the final configuration
```

6.3 Uniaxial Strain

FILE: in.phasechange

Continuing from the initialization in section 6.1, this input file uses the applied deformation of the simulation cell to reach the final system state. Simulation results from this input file are shown in figure 4. This simulation uses an NVT ensemble and applies a prescribed strain to the z-direction of the simulation box using the "`fix deform`" command to trigger the α-γRDX phase transition. Only the box dimensions are changed with "`fix deform remap none`" and the molecules and their bonds are not strained but respond to the changing environment caused by the deforming simulation box dimensions. This type of deformation isolates the strain between the molecules straddling the periodic boundary in the z-direction and each strain increment results in a displacement jump between the molecules. This displacement jump must be small so that it does not cause the boundary molecules to undergo unphysical conformational changes in response to the isolated deformation. An alternative is to use "`fix deform remap x`", where the deformation would be applied to all the atoms resulting in small stretches to bonds in molecules. This would increase the bond energy but would also distribute the deformation over the entire simulation cell. Both methods will probably work if the strain increments are sufficiently small and the system is given adequate time to equilibrate between increments. Ideally, the strain deformation should be applied by stretching the simulation box and then remapping all molecules by their center of mass position, as was done by Munday (*4*, *5*), but that cannot be done internally by LAMMPS and requires a separate post-processing step between strain increments.

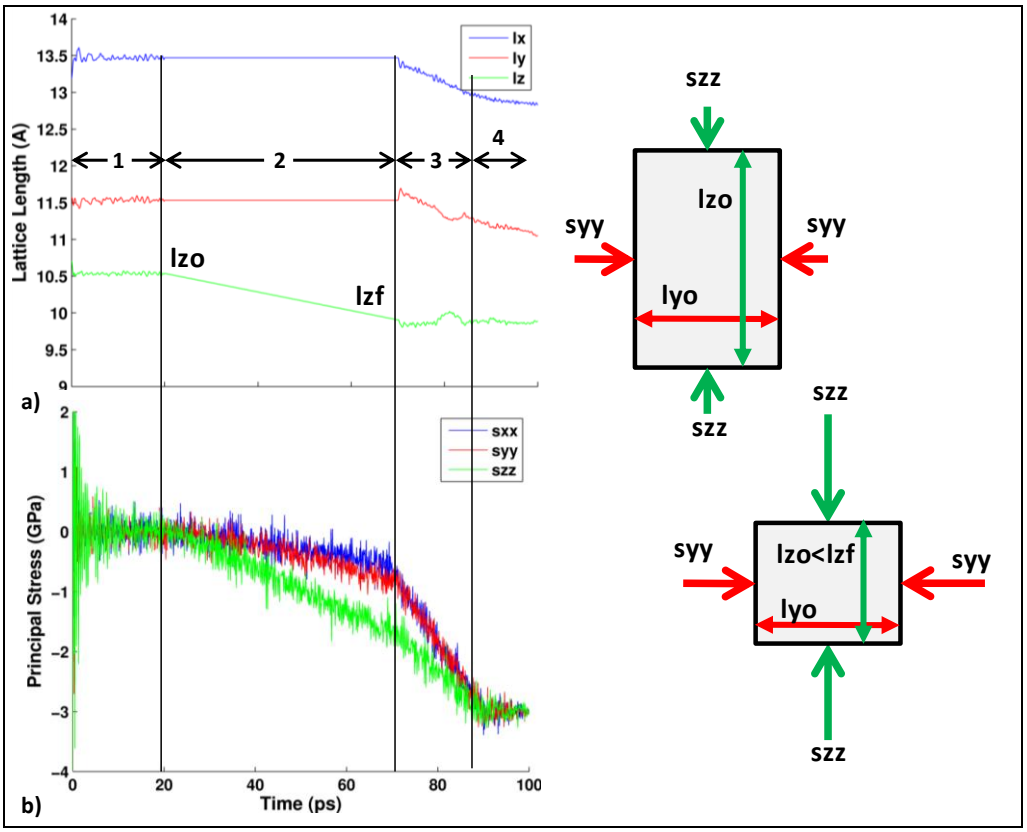

Figure 4. Uniaxial strain results using file in.phasechange. (a) Unit cell lattice constants (Å) and (b) principal stresses (GPa) vs. simulation time. The pictures indicate the applied loads and deformation. Regions on plots are labeled by the steps 1–4 in given in the in.phasechange file.

In figure 4, the following steps given in the in.phasechange file are illustrated:

- Step 1: For 20 ps, NsT equilibrate at P = 0 GPa, T = 300 K with lattice vectors held orthogonal.
- Step 2: For 50 ps, apply ramped up uniaxial strain in the z-direction to 0.94 the original z lattice length.
- Step 3: For 20 ps, ramp the stresses from the final stress state at the end of step 3 to the isotropic state of stress given by `${szz}`=P=3 GPa.
- Step 4: Langevin quenching using Nsh ensemble (constant orthotropic stress, constant energy), presented in section 6.5. Quenching under the Nsh ensemble allows the lattice to thermally contract.

The simulation is done in four steps. In Step 1, the experimental structure is equilibrated at P = 0 GPa and T = 300 K in the NPT ensemble with the "`aniso`" keyword allowing the lattice vectors to fluctuate independently while their angles relative to one another are held fixed. In Step 2, the strain is applied with "`fix deform z scale`" command. In this input script, the amount of scaling is hard coded to shrink the z dimension of the simulation cell by 0.94 or the

equivalent of ~6% strain. This strain applied to the [010] lattice direction was found by Munday (*4, 5*) to be sufficient to cause the α-γRDX phase transition. In Step 3, an NsT ensemble is used to ramp the normal stresses in the x,y,z direction from their final stress state at the end of step 2 (NVT simulation), using the "`variable sz equal pzz`" commands, to the desired isotropic stress state given by the input variable "s_zz", the variable passed in from the control file. The purpose of this script is to equilibrate the γRDX phase at an isotropic state of stress. The final step 4 is the Langevin quenching for 10 ps from T = 300 to 0 K.

This script can easily be used to produce γRDX at different isotropic states of pressure, $s_zz. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
#### NPT THERMALIZE AND COMPRESS
timestep    1.0
velocity    all create 300.0 2349851 dist gaussian

# STEP1: Thermalize at 300K and 0GPa with orthorhombic lattice vectors
dump_modify 1 first yes
fix         1 all npt temp 300 300 100 aniso 0 0 1000
run         20000
unfix       1

# STEP2: Use an NVT ensemble to maintain T=300K. Apply 0.6% uniaxial strain
#deformation to only the z-dimension of the simulation cell size over the
#length of the simulation. The atomic positions are not moved by this fix
#(remap none) and only respond to the change in the box dimensions. Box
#deformations only occur every 100 steps to allow a small amount of
#equilibration at each deformation increment. The applied strain is used to
#trigger the phase transition to gRDX. The dump rate is increased to catch
#the transition.
dump_modify 1 every 1000
fix         1 all nvt temp 300 300 100
fix         2 all deform 100 z scale 0.94 remap none units box
run         50000
unfix       1
unfix       2

# STEP3: Equilibrate in the orthorhombic NPT ensemble to the desired
#pressure. Variables are set to the final stress state of the prescribed
#strain simulation and then ramped to the prescribed pressure. In this
#simulation the variable s_zz passed in by the command line is applied as the
#system pressure and is converted from GPa to atmospheres.
variable PRESS equal ${s_zz}*9869.23267
variable sx equal pxx
variable sy equal pyy
variable sz equal pzz
fix         1 all npt temp 300 300 100 &
             x ${sx} ${PRESS} 1000 &
             y ${sy} ${PRESS} 1000 &
             z ${sz} ${PRESS} 1000
run         20000
unfix       1
```

6.4 Uniaxial Strain with Stress Averaging

FILE: in.yphasechange

This input file is a slight variation of the input file presented in section 6.3, where the average stress over a portion of the NVT simulation (step 2) is now used as the starting stress state of the NPT simulation (step 3). Simulation results from this input file are shown in figure 5. Again, the purpose of this input file is also to trigger the α-γRDX phase transition using uniaxial strain. This input file also assumes the crystal is oriented in the simulation box so that the [010] lattice vector is oriented along the y-axis and the uniaxial strain applied during step 2 is done in the y-direction. The jump in the stress state between steps 2 and 3 is reduced here by finding a time average stress state from the end of the NVT run using the command "`fix all ave/time`".

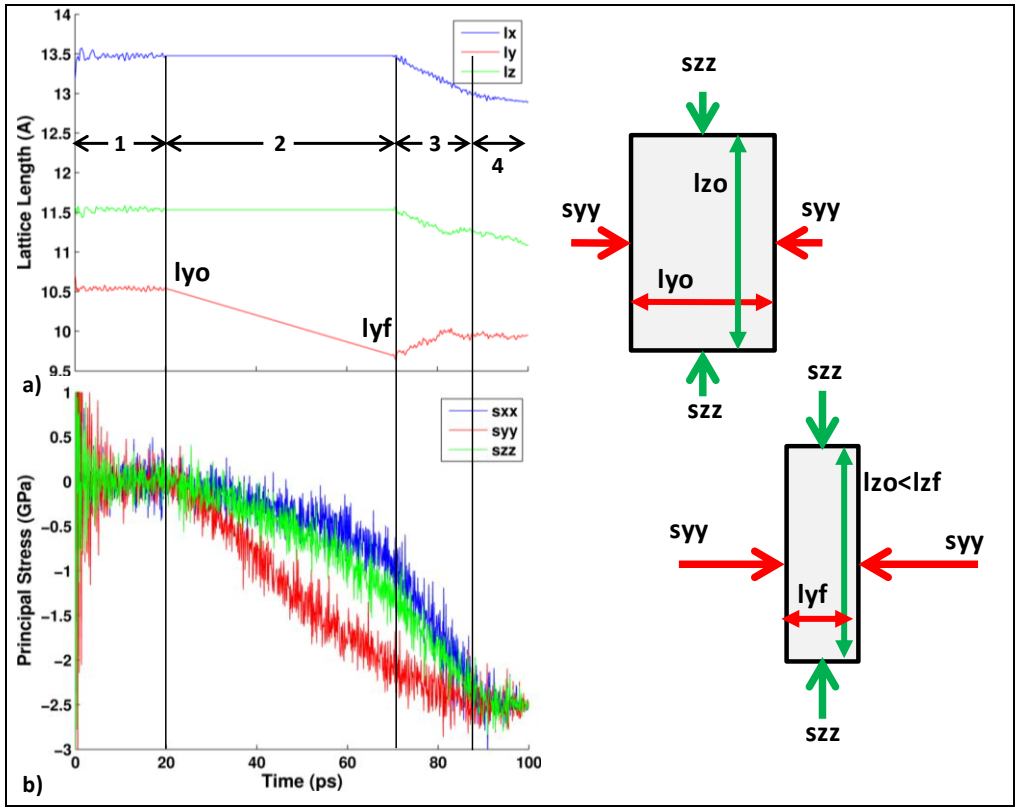

Figure 5. Uniaxial strain results using file in.yphasechange. (a) Unit cell lattice constants (Å) and (b) principal stresses (GPa) vs. simulation time. The pictures indicate the applied loads and deformation. Regions on plots are labeled by the steps 1–4 in given in the in.yphasechange file.

In figure 5, the following steps given in the in.yphasechange file are illustrated:

- Step 1: For 20 ps, NsT equilibrate at P = 0 GPa, T = 300 K with lattice vectors held orthogonal.
- Step 2: For 50 ps, apply ramped up uniaxial strain in the y-direction to 0.92 the original y-lattice length. Perform time averaging of the stress field over the final 10 ps of this portion of the simulation.

- Step 3: For 20 ps, ramp the stresses from the time averaged stress state at the end of step 3 to the isotropic state of stress given by `${szz}` = P = 2.5 GPa.
- Step 4: Langevin quenching using Nsh ensemble (constant orthotropic stress, constant energy), presented in section 6.5. Quenching under the Nsh ensemble allows the lattice to thermally contract.

This script also produces γRDX at different isotropic states of pressure, ${s_zz}. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
#### NPT THERMALIZE AND COMPRESS
timestep    1.0
velocity    all create 300.0 2349851 dist gaussian

# STEP1: Thermalize at 300K and 0GPa with orthorhombic lattice vectors
dump_modify 1 first yes
fix         1 all npt temp 300 300 100 aniso 0 0 1000
run         20000
unfix       1

# STEP2: Use an NVT ensemble to maintain T=300K. Apply 0.8% uniaxial strain
#deformation to only the y-dimension of the simulation cell size over the
#length of the simulation. The atomic positions are not moved by this fix
#(remap none) and only respond to the change in the box dimensions. Box
#deformations only occur every 100 steps to allow a small amount of
#equilibration at each deformation increment. The applied strain is used to
#trigger the phase transition to gRDX. The dump rate is increased to catch
#the transition. Time averaging using fix ave/time is used to average the
#volumetric stress components for the final 10000 timesteps of the
#simulation. The initial equilibration is over 20000 steps to capture the
#final 10000 steps the fix ave/time start 60000 is used where the timesteps
#are cumulative over an entire simulation.
dump_modify 1 every 1000
fix         1 all nvt temp 300 300 100
fix         2 all deform 100 y scale 0.92 remap none units box
variable sx equal pxx
variable sy equal pyy
variable sz equal pzz
fix         3 all ave/time 10 100 1000 v_sx v_sy v_sz start 60000
run         50000
unfix       1
unfix       2

# STEP3: Equilibrate in the orthorhombic NPT ensemble to the desired
#pressure. Variables are set to the averaged stress state from the
#prescribed strain simulation and then ramped to the prescribed pressure. In
#this simulation the variable s_zz passed in by the command line is applied
#as the system pressure and is converted from GPa to atmospheres.
variable tmp equal f_3[1]
variable sx equal ${tmp}
variable tmp equal f_3[2]
variable sy equal ${tmp}
variable tmp equal f_3[3]
```

```
variable sz equal ${tmp}
unfix       3
variable PRESS equal ${s_zz}*9869.23267
fix         1 all npt temp 300 300 100 &
             x ${sx} ${PRESS} 1000 &
             y ${sy} ${PRESS} 1000 &
             z ${sz} ${PRESS} 1000
run         20000
unfix       1
```

6.5 Langevin Quenching

FILE: continuation of in.bulk, in.phasechange, in.yphasechange

After initialization in section 6.1 and equilibration to the desired stress or strain state using one of the input files in sections 6.2–6.4, the structure is quenched to T = 0 K. This is done over the final 10 ps of the simulations shown in figures 3b, 4b, and 5b where the stress oscillations decrease. Quenching from T = 300 K to T = 0 K is done using a Langevin thermostat to bring the temperature down. The Langevin thermostat adds a frictional force term to damp out energy from the system and a random force term proportional to the desired temperature that mimics a heat bath. These force terms are in addition to the force terms calculated from the atomic interactions given by the potential energy. The Langevin thermostat does not integrate the equations of motion and must be used with an ensemble that does not control the temperature like the NVE or NPh ensemble. The temperature control comes from the Langevin thermostat. The NPh ensemble is used here to allow the simulation box dimensions to contract as the temperature is reduced. Using more steps or a different frictional drag term does not drastically affect the final energy or configuration, meaning a similar minimum energy configuration is reached. The Nph ensemble allows the simulation cell to thermally contract as the temperature is decreased as seen by the negative slope for all lattice constants in figures 3a–5a. The stress level is also maintained during quenching and the oscillations are reduced as the thermal vibrations decrease at the lower temperatures in figures 3b–5b. This procedure is useful for quenching any structure, because it allows rigid bodies to be included in the minimization. Other faster frictional damping style minimizations such as "`quickmin`" and "`quickfire`" can be used if no rigid bodies are present in the model. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
#### LANGEVIN COOL DOWN
#convert stress (GPa) to atmospheres
variable tmp equal ${s_zz}*9869.23267
#Apply an NPh ensemble to control the stress state as the Langevin thermostat
#decreases the temperature from 300 to 0 over the 10000 step run, 100.0 is a
#frictional damping parameter. Another random force due to a pretend solvent
#at the specified temperature is also added. It is hoped that quenching the
#simulation this way will allow for better relaxation of the degrees of
#freedom.
fix         1 all nph x 0 0 1000 y 0 0 1000 &
        z ${tmp} ${tmp} 1000 couple none nreset 1000
```

```
fix         2 all langevin 300.0 0.0 100.0 699483
run         10000
unfix       1
unfix       2
```

6.6 Minimization with Box Relaxation

FILE: continuation of in.bulk

After the structure has been quenched, it is minimized under alternating constant volume and relaxed volume. The manual should be consulted to determine if these are the appropriate minimization parameters for a particular problem. Due to the complex potential and number of degrees of freedom in this simulation, the system must be close to its minimized state before minimizing. This reduces the chance of the minimization finding a local minimum. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
##### MINIMIZE STRUCUTRE
### MINIMIZATION INPUT VARIABLES
variable etol equal 0.0
variable ftol equal 1.0e-6
variable maxiter equal 1000
variable maxeval equal 10000
variable dmax equal 1.0e-2

# STEP1: Choose minimization parameters.
#Set thermo 0 so that only the final minization data is output. Make sure
#velocities are reset to 0. Using steepest descent (sd) minimization where
#search direction is the force direction. It is slower than conjugate
#gradient but is supposed to be more robust which is important because there
#are so many DOFs to be minimized in the molecule. For min_modify, dmax is
#the maximum distance an atom can move per minimization loop. Min_modify
#line backtrack is the line search method and determines the minimum when the
#absolute change in energy between iterations is small enough.
thermo      0
velocity    all create 0.0 2349851
min_style   sd
min_modify  dmax ${dmax} line backtrack

#STEP2: Volume relaxation minimization
# fix box/relax allows the simulation cell to relax during minization by
using Parinello and Rahman algorithms that minimize the strain energy. The
fix box/relax x y and z setting are the applied stresses. vmax is the amount
the volume can change per minimization iteration. The reset command can be
used if change in the box compared to the initial configuration is expected
to be large.
fix         1 all box/relax x 0 y 0 z ${tmp} couple none vmax 0.001
minimize    ${etol} ${ftol} ${maxiter} ${maxeval}
unfix       1
#STEP3: Fixed volume minization
minimize    ${etol} ${ftol} ${maxiter} ${maxeval}
#STEP4: Volume relaxation minimization
fix         1 all box/relax x 0 y 0 z ${tmp} couple none vmax 0.001
minimize    ${etol} ${ftol} ${maxiter} ${maxeval}
```

```
unfix       1
#STEP5: Final fixed volume minization
minimize    ${etol} ${ftol} ${maxiter} ${maxeval}
#STEP6: Create restart file so that other simulation conditions can be run
from the minized configuration.
write_restart     restart.min
```

6.7 Matlab Script for Post-Processing

FILE: read_log.m

A generic Matlab script is given here to quickly read in thermo data from a LAMMPS simulation and perform post-processing. The following Matlab file was used to produce figures 3–5. The Matlab function textscanlog.m described in section 6.8 is used to parse the thermo data and save them into a cell structure that can be plotted by the main program. This function is a faster version of the function readlog.m that comes with the LAMMPS distribution in the folder /tools/matlab. Each LAMMPS "`run`" command produces a separate set of thermo data that are saved into an individual cell. In this Matlab script, all of the thermo output data from each run are concatenated into a single array, making it easier to plot the entire time series. This Matlab script is located in the folder /RDX/aMinimize. Other commands can be added to this script to do averaging over certain sections of the simulation (each stored in their own cell) to determine stress/strain relations. In the Matlab file shown below, the `%` and green letters indicate comments and are skipped over by Matlab.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
% Read out LAMMPS log file written by thermo command    %
% Use readout.m function included with LAMMPS distribution %
% Lynn Munday                          %
% Started March 25, 2012                    %
%                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all hidden,clear all, clc
%
% MANUAL INPUT
lammps_path='/usr/people/lmunday/Lammps/lammps-21Dec11/data/RDX/aMinimize';
addpath(['/usr/people/lmunday/Lammps/zMATLAB/tools/matlab'])
%
file_in=[lammps_path, '/RESULTS.010/RESULTS.yphaseb2szz2.50/log.lammps']
logdata = textscanlog(file_in); %call function to function
%
```

```
%concatenate separate runs into single array
B=[];
for i=1:size(logdata.Chead,1)
 B=vertcat(B, cell2mat(logdata.data(i,:)) );
end
% print out recorded data
fprintf('Number of simulation fixes: %d \n',size(logdata.Chead,1))
for i=1:size(logdata.Chead,2)-1
 fprintf('%5d  %s \n',i,logdata.Chead{1,i})
end
%PLOT TIME SERIES
figure(1)
axes('FontWeight','bold','FontSize',12);
hold on
 plot(B(:,1)/1000,B(:,11)*0.000101325,'-b')
grid on
title('Statis File', 'fontsize',14)
xlabel('Time(ps)', 'fontsize',14)
ylabel('Pressure (GPa)', 'fontsize',14)
%
figure(2)
axes('FontWeight','bold','FontSize',12);
hold on
 plot( B(:,1) ,-B(:,12)*0.000101325,'-b')
 plot( B(:,1) ,-B(:,13)*0.000101325,'-r')
 plot( B(:,1) ,-B(:,14)*0.000101325,'-g')
grid on
legend('sxx','syy','szz')
title('Statis File', 'fontsize',14)
xlabel('steps', 'fontsize',14)
ylabel('Principal Stress (GPa)', 'fontsize',14)
```

6.8 Matlab Function for Parsing LAMMPS Log Files

FILE: textscanlog.m

The following is the Matlab function used to parse the LAMMPS log file. This function speeds up the parsing data by 5 times over the function readlog.m that is distributed with the LAMMPS software in the folder /tools/matlab. The function textscanlog.m offers a larger speedup as the amount of thermo data collected per run increases. This function outputs the data in a similar structure as readlog.m with one cell containing the logfile headers and another cell containing the data. Inside the data cell, the data for each run are stored in a single row of cells and each cell in the row contains a column of output thermo data for each step of the run.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = textscanlog(varargin)
% Read LAMMPS log files
% input is log file name with path
% output is a structure --> out
% out.Chead --> has heading of columns in each run
% out.data --> has the data from each run sorted into a cell for each column
```

```
% Type cell2mat(out.data(i)) to get the numeric array
%
% Example
%    logdata = textscanlog('log.LAMMPS');
%
logfile = varargin{1};
try
  fid = fopen(logfile,'r');
catch
  error('Log file not found!');
end

chdr = 1;
while ~feof(fid)
 s = fgetl(fid);
 if length(s) > 4 && strcmp(s(1:4),'Step')
  header(chdr,:) = textscan(s,'%s');
  [a,b] = textscan(fid,repmat('%f ',1,size(header{chdr,:},1)));
   store(chdr,:) = a;
   chdr = chdr+1;
 end
end
fclose(fid);
%--------OUTPUT-----------------------------------------
out.Chead = header;
out.data = store;

varargout{1} = out;
%
```

6.9 PBS Script for Submitting Jobs

FILE: qsub.Pmin

This script is used to submit simulations to the batch queuing system, pbs, on the ARL cluster Harold. The top lines containing "`#PBS -…`" are not commented out but are pbs commands. The other top lines in blue with "`#`" followed by a space are comments and are skipped over by pbs. More information on the pbs commands used below is available at the ARL DSRC Web site http://www.arl.hpc.mil/docs/pbsUserGuide.html. The ARL Web site also gives several pbs commands used for killing and holding batch jobs, checking the status of jobs, and the number of jobs waiting on the queue.

The equilibration and minimization simulations given in this section require about 100,000 integration steps to apply the loads, equilibrate the system, and quench the system followed by several more minimization cycles. These simulations containing 3x3x3 unit cells of RDX (4536 atoms) take ~2 h, "`#PBS -l walltime=2:20:00`" using 16 processors "`#PBS -l select=2:ncpus=8:mpiprocs=8`". An extra 20 min of wallclock time is requested to ensure that the simulation is completed. Asking for too much extra time will delay the start of the simulation in the queuing system. The requested jobname, (`#PBS -N lmp_010b2P2.50`) is used later in the script to identify the configuration of the unit cell to be used such as the z-

direction of the lattice vector, unit cell layer interface (b1 or b2), and the value of the variable "s_zz" to be passed into the LAMMPS script during execution. This makes it possible to easily direct input/output and alter simulations by changing only a single line of the pbs file, "#PBS -N".

The rest of the pbs file is a basic CSH script to direct input and output of the simulation. A pbs file is submitted to the queue in the command prompt by typing "> qsub qsub.Pmin". However, the pbs commands are not executed until the queuing system submits the job for execution. This means that there is a wait time between when the file is submitted and when it will be executed. This is normally less than an hour for the small simulations run in this section. For larger simulations using 1000's of processors and running for tens of hours, the wait time will be days. During the wait time, any changes made to the LAMMPS input file will be incorporated into the executed simulation. Changes made to the pbs file will not be incorporated into the execution and the same pbs file can be slightly modified to submit several simulations. For instance, in this file, a range of "szz" = 0.00, 0.25 0.50, 1.50, etc., can be submitted by only changing the line "#PBS -N" without having to create separate LAMMPS input files for each "szz". In the pbs script shown below, the # and blue letters indicate comments and are skipped over.

```
#!/bin/csh
# Request maximum wallclock time for the job
#PBS -l walltime=2:20:00
# select=number_of_nodes,ncpus=cores/node,mpiprocs=MPI_procs/node
# Total cores requested = number_of_nodes X MPI_procs/node
#PBS -l select=2:ncpus=8:mpiprocs=8
# Request job name
#PBS -N lmp_010b2P2.50
# Request the PBS job queue for the job
#PBS -q standard
# Specify how to distribute MPI processes across nodes
#PBS -l place=scatter:excl
# Combine the output and error files into a single file
#PBS -j oe
# Select Project ID
#PBS -A PROJECTID###
# Request environment variables be exported from the script
#PBS -V
#
############################
# script for LAMMPS
############################
#
# Get name of tmp folder where simulation will be run
set JOBID=`echo "$PBS_JOBID" | cut -f1 -d[`
#
# Setting JOBID to use job number PBS has JOBID[#].o2
set IN=in.yphasechange #input file name
# cut-up qsub name into variables
set DIR=`echo "${PBS_JOBNAME}" | cut -c5-7`
set BASIS=`echo "${PBS_JOBNAME}" | cut -c8-9`
```

```
set szz=`echo "${PBS_JOBNAME}" | cut -c11-14`
# Print data to qsub log file
echo "PBS_JOBNAME=${PBS_JOBNAME}"
echo "started job ${JOBID} on" `date`
echo "GSF plane normal: DIR=${DIR}"
echo "GSF plane BASIS=${BASIS}"
echo "Uniaxial stress szz=${szz} GPA"
echo "lammps input file name IN=${IN}"
# Set LAMMPS file paths in user directory
set HOMEDIR=/usr/people/lmunday/Lammps/lammps-21Dec11/data/RDX
set DATADIR=${HOMEDIR}/aMinimize/RESULTS.${DIR}
set RESDIR=${DATADIR}/RESULTS.yphase${BASIS}szz${szz}
echo HOMEDIR is ${HOMEDIR}
echo DATADIR is ${DATADIR}
echo RESDIR is ${RESDIR}
# Go to tmp folder where simulation will be run
set TMPD=/usr/var/tmp/lmunday/${JOBID}
mkdir -p ${TMPD}
echo TMPD is ${TMPD}
# copy LAMMPS data from user directory to temporary execution directory
cp ${DATADIR}/../${IN} /${TMPD}/${IN}
cp ${HOMEDIR}/zPOTENTIALS/potential_ewaldn.mod /${TMPD}
cp ${DATADIR}/data.aRDX${BASIS}_3x3x3 /${TMPD}/data.aRDX
cd ${TMPD}
echo
echo contents of ${TMPD} before start is:
ls -l
echo
#
# LOAD MODULES FOR MPI, these were the compilers used to build the program
unlimit
module load compiler/intel11.1
module load mpi/sgi_mpi-1.26
#
echo starting program execution on `date`
# run simulation and pass in variable from jobname (s_zz)
set EXE=${HOMEDIR}/../../src/lmp_harold
mpiexec_mpt ${EXE} -v s_zz ${szz} < ${IN}
# program completed
echo completed program execution on `date`
echo
echo contents of ${TMPD} after completion is:
ls -l
echo
# tar up and copy output back to user directory
tar -cvf ../results.${JOBID}.tar *
cd ..
mkdir ${RESDIR}
cp results.${JOBID}.tar ${RESDIR}
cd ${RESDIR}
tar -xvf results.${JOBID}.tar
rm results.${JOBID}.tar
echo contents of results directory is:
ls
echo
echo completed job ${JOBID} on `date`
```

6.10 CSH Script for Executing Single LAMMPS Simulation

FILE: run.Pmin

A CSH script is presented here that matches the essential features of the pbs batch script given in section 6.8 to execute the LAMMPS simulations given in section 6. The purpose of this script is to run LAMMPS simulations on a local multicore Linux computer like the Dell computer described in the makefile given in section 4. This script copies the LAMMPS files to the correct directory to be executed by LAMMPS. The access to this script must be set to executable by using the command "`chmod 757 run.Pmin`". This must be done to execute any script and more information about the `chmod` command and its options is available by typing "`chmod --help`". The LAMMPS simulation is executed immediately upon running the script; there is not a wait time. In the CSH script shown below, the # and blue letters indicate comments and are skipped over.

```
#!/bin/csh
#
# Simple script to run a single LAMMPS simulation on local Dell workstation
#
# Set simulation inputs
set IN=in.yphasechange
set DIR=010
set BASIS=b2
set szz=2.50
#
# Set up paths for copying and writing data
set HOMEDIR=/data/lammps-27Oct11/data
set DATADIR=${HOMEDIR}/RDX/aMinimize/RESULTS.${DIR}
set RESDIR=${DATADIR}/RESULTS.yphase${BASIS}szz${szz}
rm -r ${RESDIR}
#
# Copy data to RESDIR for simulation
mkdir ${RESDIR}
cp ${DATADIR}/../${IN} /${RESDIR}/${IN}
cp ${HOMEDIR}/zPOTENTIALS/potential_ewaldn.mod /${RESDIR}
cp ${DATADIR}/data.aRDX{BASIS}_3x3x3 /${RESDIR}/data.aRDX
#
# Go to results directory for simulation
cd ${RESDIR}
#
# LOAD MPI MODULE used to build LAMMPS
module load cse/openmpi/1.4.1
# RUN SIMULATION
set EXE=/data/lammps-27Oct11/src/lmp_LynnDell
mpirun -np 8 ${EXE} -v s_zz ${szz} < ${IN}
```

7. LAMMPS Generalized Stacking Fault Simulations

FILE: in.shift

The LAMMPS input file to be presented in this section was used to run the GSF simulations in the work by Munday (*4, 6*). The GSF simulations use the LAMMPS implementation of the SB (*3*) potential file presented in section 5.1; the same potential file that was used during the equilibration/minimization simulations in section 6. The final minimized configurations from section 6 are used here as the initial structures to create the stacking faults. Details of the purpose, procedure, features, and results of the stacking fault simulations are given by Munday (*4, 6*).

Section 7.1 first presents the procedure used to create the stacking fault structures from the minimized bulk simulations. Next, the LAMMPS input file that runs the stacking faults will be presented in section 7.2. The input file is broken into the follow three sections: (1) initialization (section 7.2.1), (2) rigid molecule interface separation/decohesion (section 7.2.2), and (3) flexible molecule stacking fault (section 7.2.3). The input file runs only a single stacking fault structure configuration given by the stacking fault vector (f_x,f_y). Section 7.3 presents the pbs and CSH scripts used to submit a series of stacking fault configurations to allow f to span the entire surface area of one unit cell. The final section 7.4 presents the Matlab scripts used to post-process the results to determine the GSF energy surface using equations 1–4 from section 2 and described in more detail in references (*4-6*).

7.1 Configuration

/aMinimize/lammpstrj2data.m

The configuration presented in this section is of a very specific format that matches the LAMMPS "`group`" commands given in the LAMMPS input file in.shift to be presented in section 7.2. The LAMMPS groups are created under the assumption that molecule IDs in the data file refer to z-layers of the crystal. The details given here about molecule orientation are only necessary for the stacking fault simulations to be run in this work and are not essential requirements to stacking fault simulations in general. The initial cell used to create the stacking fault to be simulated is created from the minimized structures given by the simulations presented in section 6. This section describes a Matlab script, /aMinimize/lammpstrj2data.m, which basically replicates a dump file of the minimized crystal in the stacking fault direction, adds a vacuum layer to the top of the replicated crystal in the z-direction, renames the molecule IDs to refer to the z-layers of the crystal, and finally creates all of the appropriate bonds, angles, dihedrals, and impropers and writes it out to a LAMMPS data file.

The stacking fault structure to be created is a slab in the xy plane with a vacuum region separating the z-periodic images. The stacking faults are then created by shifting the top and bottom halves relative to one another in the xy plane of the simulation cell. This ensures periodicity of the stacking fault structure in the xy plane. The normal to the stacking fault plane must be in the z-direction of the simulation cell. Therefore, the minimized structure must be oriented with the correct lattice vector in the z-direction. In this work, several experimental structures were created with the proper orientation (stacking fault normal placed in the z-direction) and were then minimized. Alternatively, a more computationally efficient method would be to reorient a single minimized structure to the proper stacking fault structure.

After a properly oriented crystal is created with the stacking fault plane normal in the z-direction the Matlab script lammpstrj2data.m in the /aMinimize folder is used to create the stacking fault structure from the bulk minimized dump file. The Matlab script will only work with unwrapped molecules that use the LAMMPS command "`dump 1 all custom 10000 all.lammpstrj id type xu yu zu`", where "`xu, yu, zu`" indicate molecule coordinates that are not cut along the periodic boundaries. The Matlab scripts begin with hardcoded values to direct where the minimized LAMMPS dumpfile is located, the dimension of the crystal in unit cells to be read in, the number of replications of the structure in each direction, the simulation cell symmetry of the output data file (orthorhombic or triclinic), the thickness of the vacuum layer to be placed in the z-direction, and the adjustments to the interplanar spacing. For the αRDX crystal, the symmetry operators of the Pbca space group result in an interplanar spacing equal to half the unit cell length in the [100], [010], and [001] directions, as shown in figure 6. The Matlab script also includes deformation gradients that can be used to apply thermal expansion to the unit cell. The deformation gradient is applied to the center of mass position of each molecule and therefore does not artificially strain molecular bonds. Other deformation gradients can be given here to apply mechanical strains. In this work, the deformation gradient is set to the identity and no thermal expansion is applied to the minimized crystal.

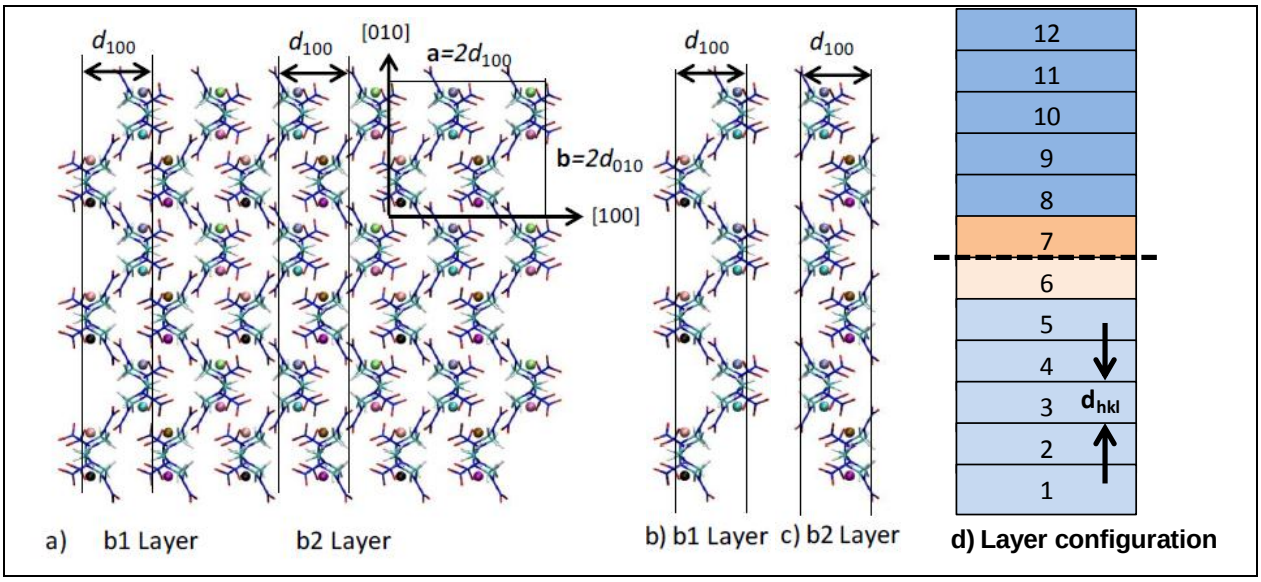

Figure 6. Layering of the αRDX crystal. (a) (001) projection of 3×3 unit cell αRDX crystal with molecules shown by their bonds and colored spheres represent each molecules center of mass position. The black lines oriented in the [100] direction indicate the [100] interplanar thickness, d_{100}. Two separate layers, labeled b1 and b2, can be formed for αRDX as indicated in a) and shown separately in (b) and (c). (d) Stacking fault configuration used for the simulation where the dashed line indicates the interface where the dark colored slabs are shifted relative to the light colored layers.

As shown in figure 6, for a screw axis, inversion center, or glide plane symmetry operator, the interplanar thickness is reduced to ½ the unit cell thickness, which is the case for the Pbca spacegroup to which αRDX belongs. The b1 layer (b) is created along the face of the unit cell provided by Choi and Prince (*2*), and the b2 layer (c) is created by shifting by ¼ unit cell in the [100] direction. The attachment energy (*4*) for a layer is found as the energy difference between the layer shown in (b) and the bulk crystal. The stacking fault is created on the interface between two layers and it was found by Munday (*4, 6*) that the layer with the lower attachment energy also had the lowest unstable stacking fault energy. Also in figure 6, in the stacking layer configuration (d), the blue layers are treated as rigid slabs of molecules and the orange slabs are fully flexible molecules. The structure contains 12 d_{hkl} layers in the z-direction of the simulation cell or 6 unit cells for the (100), (010), and (001) stacking fault planes. The simulation cell is 3-D periodic requiring a vacuum layer to be placed above layer 12 to keep it from interacting with layer 1.

The Matlab script uses the function readdump.m to read in the data from a LAMMPS dumpfile. The readdump.m function is available in the /tools/matlab directory of the LAMMPS distribution. The readdump_one command used is "`readdump_one(file_in,-1,5)`", where "`-1`" is used to read in the last configuration in the dumpfile and "`5`" indicates that five elements of data are to be read in for each atom, which include atom name, molecule name, and the "`xu,yu,zu`" unwrapped coordinates. These are indicated in the dump command issued in the minimization input files.

After reading in the dumpfile, the simulation box dimensions are recreated. These are not the same as the box dimensions of the dump file. The minimized crystal is then replicated to create the stacking fault structure of the proper size. In this work, all of the minimized cells contained 3x3x3 unit cells and the stacking faults are created by replicating this once along the z-axis (3x3x6). It was found that 6 unit cells in the z-direction was a sufficient distance to separate the stacking fault interface from the vacuum layers. The center of mass for each molecule is then found and these are used as the coordinates to divide up the molecules into their respective layers. The molecule IDs are then renumbered to correspond to the layer ID, as shown in figure 6d. Figure 6d indicates 12 layers because each layer thickness is given by the interplanar spacing where $2d_{hkl}$ = unit cell length. The interplanar spacing of skewed unit cells is equal to the full unit cell length because the symmetry conditions that shorten the interplanar spacing do not apply to the skewed axis. The bonds are then computed for the new structure and a new LAMMPS data file is created that can be used in the stacking fault simulations.

7.2 Single Stacking Fault Input File

FILE: in.shift

The single LAMMPS input file presented in this section was used to run the GSF simulations in the work by Munday (*4, 6*). The input file is broken into the three sections: (1) initialization (section 7.2.1), (2) rigid molecule interface separation/decohesion (section 7.2.2), and (3) flexible molecule stacking fault (section 7.2.3).

7.2.1 Initialization

The first section of the file in.shift initializes the simulation using the same commands presented in section 6.1.1. To maintain consistency, the stacking fault simulations should use the same potential file used to create the minimized structure used to create the stacking fault configuration presented in section 7.1. Three variables are passed into the LAMMPS script from the command line, `s_zz`, `ix` and `iy` during execution. The first two variables (`ix,iy`) are the stacking fault vector, **f**=(f_x,f_y), i.e., the xy-coordinates of the stacking fault offset across the glide plane. A single unit cell stacking fault area is divided up into a 100x100 grid and (`ix,iy`) provides the offset distance in increments on this grid, where (`ix,iy`) = (0,0) is the initial crystal, i.e., (`ix,iy`) = (50,25) is the stacking fault with a normalized stacking fault vector (f_x,f_y) = (50/100,25/100) = (0.5,0.25). The stacking faults are periodic with their period being one unit cell, i.e., (f_x,f_y) = (0,0.3) = (0,1.3) = (1,0.3), etc. The variable `s_zz` is the uniaxial stress that will be applied in the z-direction to the flexible stacking fault portion of the simulation. This stress needs to be chosen to match the z-component of the stress applied during the minimization used to create the initial structure. For most stacking fault simulations found in literature (*7*), `s_zz` = 0, which was the value used by Munday (*4, 6*). A nonzero value for `s_zz` must be used in stacking fault simulations of the high-pressure γRDX phase to keep the material compressed. Without the additional uniaxial stress, the material would expand into the vacuum region, lowering the pressure in the material possibly resulting in the γ→αRDX transition. There also

variables `nxlat`, `nylat`, and `nzlat` hardcoded to the number of unit cells contained in the input file. These are used to determine the lengths of the unit cell in each direction.

The LAMMPS "`group mobile molecule <> 6 7`" command is used to separate the crystal into layers in the z-direction. Each layer is treated differently during the simulation. The groups are created by grouping atoms together by their "molecule" ID. The configurations created in section 7.1 were specifically created with the molecule IDs were set to the layer numbers, as shown in figure 6. The "`mobile`" group consists of all inclusive layers between layers 6 and 7 "`<> 6 7`". The "`mobile`" layer is shown by shades of orange in figure 6 and by the yellow and purple layers in figure 7d. The atoms "`mobile`" group will be integrated with fully flexible molecules during flexible stacking fault simulation. The "`top`" group contains layers 7 thru 12 and the "`bottom`" group contains layers 1 thru 6. Molecules can belong to multiple groups. The plane containing the stacking fault is created between layers 6 and 7 or between the "`top`" and "`bottom`" groups. The group "`dout`" is used to control which atom coordinates are written out during dump files. Only the atoms in the four layers surrounding the stacking fault are written out, as shown in figure 7.

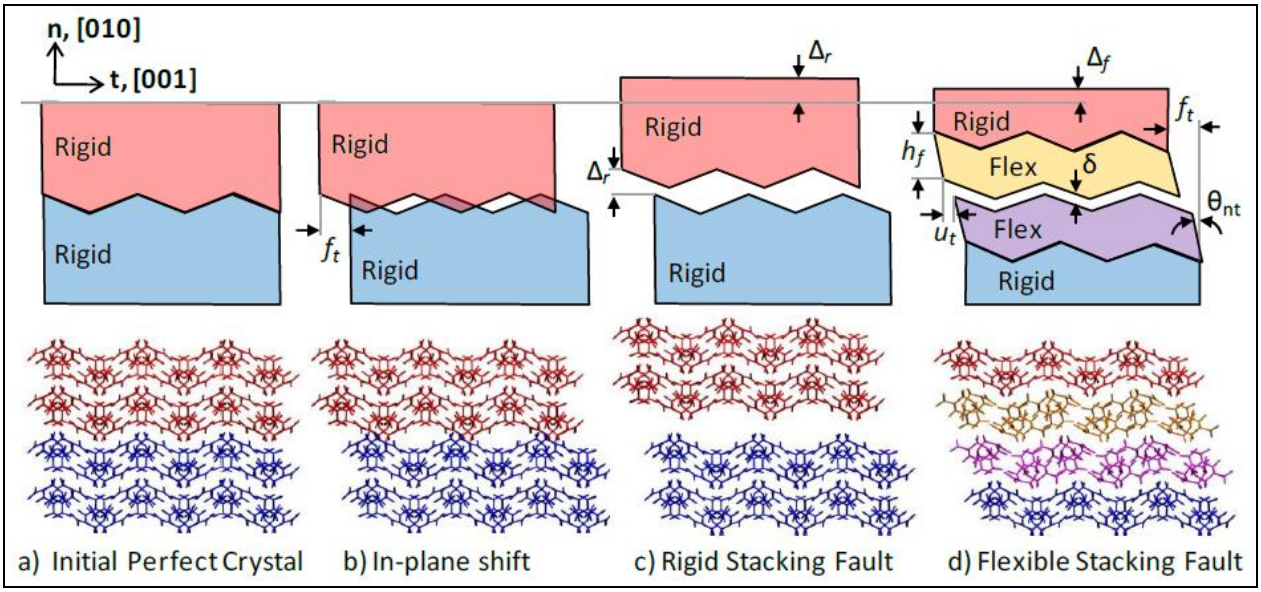

Figure 7. Sequence of steps used to create a single quenched b2 (010) stacking fault in the [001] slip direction, with a crystal lattice only showing two unit cell layers in the [010] direction of the actual six used in the simulations. (a) Initial minimized perfect crystal structure; (b) creation of the initial rigid stacking fault structure with the top (red) and bottom (blue) halves offset by the stacking fault vector f_t; (c) the rigid stacking fault structure created by separating the interface in the [010] direction by Δ_r to reduce the repulsive force caused by overlap; and (d) the flexible stacking fault structure, where a single layer of molecules with thickness d_{010} above and below the stacking fault, is allowed to relax as fully flexible molecules.

Figure 7 starts with the initial minimized perfect crystal structure created using the steps given in section 6. Next, we see that an overlap is caused by shifting the wavy interface, which leads to a high energy configuration. Following that, the rigid stacking fault structure is created by

separating the interface in the [010] direction by Δ_r to reduce the repulsive force caused by overlap. Δ_r is chosen as the equilibrium separation where the repulsive forces are balanced by the attractive forces. The volume of the crystal increased by an amount proportional to Δ_r. Finally, the flexible stacking fault structure, where a single layer of molecules with thickness d_{010} above and below the stacking fault, is allowed to relax as fully flexible molecules. The bottom rigid slab in blue is held fixed while the top rigid slab in red is allowed to move in the [010] direction. The flexible molecules are able to shear and change conformation to reduce energy, leading to an interfacial displacement u_t, where $u_t \neq f_t$. The interface also closes to δ and the top rigid surface reduces to Δ_f with $\Delta_f \neq \delta$. The infinitesimal shear strain of the flexible layers is given by $\theta_{nt}/2$.

At this point, four LAMMPS computes are issued. Data calculated from computes are accessible by using "`c_ID`", where ID is the name given to the compute. The first compute with ID=`1` is "`compute 1 all com/molecule`", which computes the (x,y,z) coordinates of each molecule center of mass. In this simulation, the molecules are actually layers of molecules and the COM refers to the location of the layers COM. The COM is then used during post-processing to determine the relative displacement of each layer relative to one another to compute strains and interfacial displacements. The next compute with ID=`mobtemp` is "`compute mobtemp mobile temp`", which computes the temperature from the kinetic energy associated with only the atoms contained in the flexible region of the simulation. The degrees of freedom of the frozen layers will dilute the temperature computed by LAMMPS and this provides a fix. The next compute with ID=`2` is "`compute 2 mobile pair/local dist`", which computes the distance between each atom-atom pair involved in the nonbonded interactions. This compute will only find distances in the pair list and does not include atoms bonded with bonds, angles, or improper dihedrals. The final compute with ID=3 is "`compute 3 mobile reduce min c_2`", which is used to determine the minimum nonbonded pair interaction distance. Compute ID=`3` works by computing the minimum of the pair distances given by compute ID=`2` referenced by "`c_2`". In the perfect crystal, "`c_3`" gives the minimum distance between nonbonded atoms.

The "`thermo_style`" command is then used to output the usual thermodynamic data plus the data created by the computes. The compute "`c_1`" creates a 12x3 array, where 12 is for each molecule ID and 3 is for the (x,y,z) coordinates. The COM data from "`c_1`" are written out for the six layers surrounding the stacking fault. A static simulation is then run with zero steps using "`run 0`". This initializes all of the thermo and compute variables so that they can be used in the LAMMPS variables. Variables are then created to determine the size of the crystal and calculate the size of the stacking fault. Defining the variables using two steps:

```
variable tmp equal lx

variable lx0 equal ${tmp}
```

These steps create a static variable that does not change during the simulation. This is useful for creating variables that correspond to the initial crystal structure. The unit cell dimensions are calculated here by `lx0`, `ly0`, and `lz0`, which are then used to used with the input variables

(ix,iy) to calculate the offset of the stacking fault vector in angstroms (dx,dy). Calculating these values here makes the input file slightly more complicated but more general to other crystal orientations. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
### RDX GENERALIZED STACKING FAULT ENERGY w/ applied force
### Creates a single stacking fault at ix and iy from input script
### Determines rigid seperation distance
### Molecule number is by half layers with bottom layer=1 upto 2*nlat
### SF assumes 100 increments (ninc) across unit cell (0.01*ix*xlat)

###-----------START INPUTS--------------------------###
#
### STACKING FAULT VECTOR FROM INPUT SCRIPT
print "set on command line ix=${ix}"    #offset in x-dir from origin
print "set on command line iy=${iy}"    #offset in y-dir from origin
print "set on command line szz=-${szz}GPA" #z uniaxial comp on rigid body
variable ninc equal 100                  #z uniaxial comp on rigid body
### Hardcoded supercell parameters
variable nzlat equal 6  #make sure this matches the read_data file
variable nxlat equal 3  #most cross sections are 3x3
variable nylat equal 3


### CREATE RDX CRYSTAL
dimension   3
newton      on   # always on
boundary    p p p  # periodic in xyz
units       real  # kcal/mol, fs, atm
atom_style  full  # coulomb and intra bonds
read_data   data.nz6
include     potential_ewaldn.mod

### GROUPING, LAYERS TOGETHER
# assumes molecules in configuration file were numbered by their layer
group mobile molecule <> 6 7 # Center Layers (fully flexible region)
group top molecule <> 7 12  # Top Layers (floating rigid slab)
group bot molecule <> 1 6   # Bottom Layers (displaced frozen slab)
group dout molecule <> 5 8  # Layers to DUMP

compute     1 all com/molecule
compute     mobtemp mobile temp
compute     2 mobile pair/local dist
compute     3 mobile reduce min c_2

### Setup neighbor style
neighbor    1.0 bin
neigh_modify      every 1 delay 0 check yes

### Setup output
thermo      10
thermo_style      custom step dt c_3 c_mobtemp &
                  vol lx ly lz xy xz yz &
            press pxx pyy pzz pxy pxz pyz pe ke enthalpy &
```

```
                    evdwl ecoul epair ebond eangle edihed eimp emol elong &
                    c_1[4][1] c_1[4][2] c_1[4][3] &
                    c_1[5][1] c_1[5][2] c_1[5][3] &
                    c_1[6][1] c_1[6][2] c_1[6][3] &
                    c_1[7][1] c_1[7][2] c_1[7][3] &
                    c_1[8][1] c_1[8][2] c_1[8][3] &
                    c_1[9][1] c_1[9][2] c_1[9][3]

thermo_modify       norm no

#----- FIND LATTICE SIZE AND OFFSETS FOR X AND Y -------------
run 0
variable tmp equal lx
variable lx0 equal ${tmp}
variable xlat equal ${tmp}/${nxlat}
variable tmp equal ly
variable ly0 equal ${tmp}
variable ylat equal ${tmp}/${nylat}
variable dx equal ${ix}*(${xlat})/${ninc} #SF vec in x-dir
variable dy equal ${iy}*(${ylat})/${ninc} #SF vec in y-dir
```

7.2.2 Rigid Lattice Decohesion

After initialization, a rigid stacking fault simulation is performed to determine the equilibrium separation distance across the interface for the prescribed stacking fault vector, **f**. This process is shown in figures 7b and c, where first the top half is shifted relative to the bottom half by the input stacking fault vector **f** = (`ix,iy`) leading to molecule overlap. The interface is then separated by Δ_r to reduce the system energy. These energy-separation curves are shown in figure 8. Each line corresponds to a different stacking fault vector shown in the legend. This input script runs only a single stacking fault and therefore provides only a single energy separation curve in figure 8.

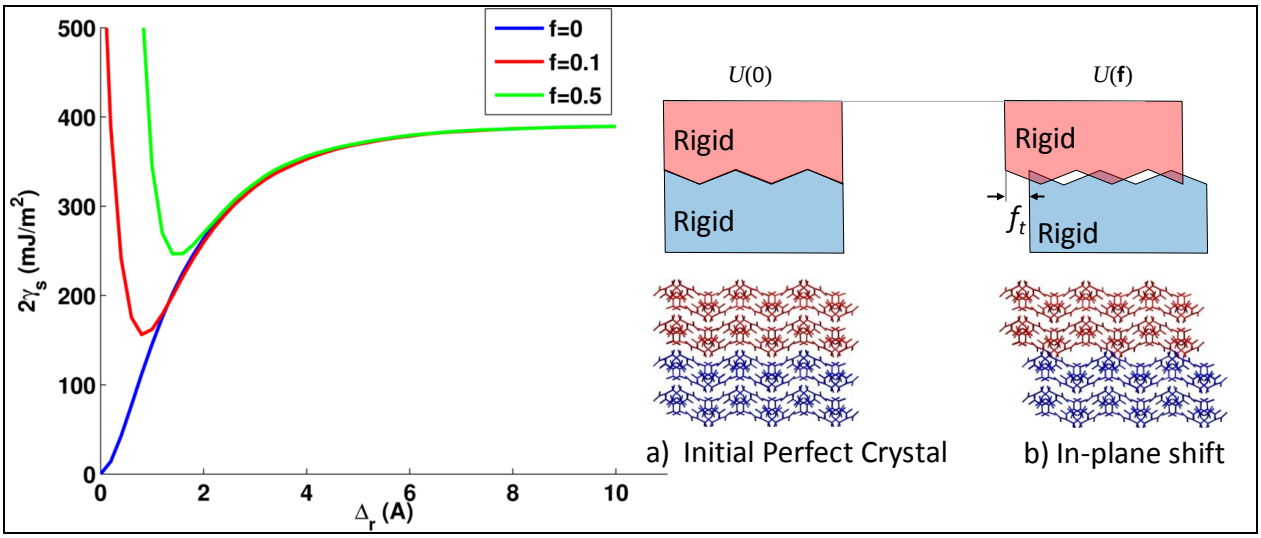

Figure 8. Energy separation curves for rigid lattice stacking faults. (a) The perfect crystal corresponds to the blue curve, which starts at 0 and increases to free surface energy. (b) Initial configuration for stacking fault f_t=0.5. Energy separation curve for this configuration is shown by the green line.

In figure 8, for $\Delta_r = 0$, the green line goes to extremely high values due to the large repulsive interactions. The energy is then reduced as the interfaces are separated. The minimum energy separation distance is given for Δ_r ~1.8 A. Continuing to separate the interface causes the energy to again increase until the free surface energy is reached. For a given stacking fault interface, the same free surface energy is given for all stacking fault vectors f.

The rigid stacking fault portion of in.shift begins by offsetting the "`top`" group relative to the "`bottom`" group by the stacking fault vectors, (`${dx} ${dy}`) and also separating the top from bottom by 10 Å to create a crystal with two free surfaces by using the following LAMMPS command:

"`displace_atoms top move ${dx} ${dy} ${iz} units box`"

The "`displace_atoms`" command moves the atoms contained in the group "`top`" by the offset amount (x,y,z)=(`${dx},${dy},${iz}`). The offset "`units box`" refers to the units of the simulation, which are Å. A static simulation is then done, "`run 0`", to execute the displacement. Some initial variables, "`miniz`" and "`minPE`", are then calculated to help determine the minimum energy separation distance. A LAMMPS loop is then done that will perform a series of "`displace_atoms`" and static simulations, "`run 0`", that will incrementally lower the "`top`" group by "`dz`" = –0.2 Å from 10 to 0 Å. At each step, an "`if`" statement is used to compare the current potential energy to the minimum potential energy found at other separation distances. If the current potential energy is lower it is saved as the minimum energy, "`minPE`", and "`miniz`" is then set at the current separation distance. The "`if`" statement also checks that no nonbonded atoms are placed closer than 1 Å during the "`displace_atoms`" command. The SB potential uses the Buckingham potential to calculate

nonbonded interactions. Under normal simulation conditions, the Buckingham potential is sufficiently repulsive as to not allow molecules to come to close together. However, for unphysically small atom separations (<1Å), the r^{-6} dispersion term in the Buckingham potential goes to zero faster than the short range exponential repulsion term. This causes the atoms to become extremely attractive, leading to very negative potential energies. The wavy interface of the stacking fault leads to this type small molecule overlap, as shown in figure 8b, where nonbonded atoms become very close to one another leading to very small potential energies. However, these configurations are tracked by the "`if`" statement and the computes given by "`c_2`" and "`c_3`" not included in "`minPE`" and "`miniz`". The current crystal configuration at the end of the "`loopa`" is shown in figure 8b, where **f** = (`ix, iy`) and $\Delta_r = 0$. The LAMMPS input file in.rigid runs only to this point of the simulation and does not proceed with the flexible molecule quenching portion of the simulation, which is more computationally expensive. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
#----- FIND MINIMUM ENERGY RIGID SEPARATION ------------------
variable iz equal 10.0
displace_atoms top move ${dx} ${dy} ${iz} units box
run 0
variable miniz equal ${iz}
variable tmp equal pe
variable minPE equal ${tmp} #initially set to free surf PE

# loop to find minimum energy
variable dz equal -0.2
label loopa
variable a loop 50
 variable tmp equal ${iz}+${dz}
 variable iz equal ${tmp}
 displace_atoms top move 0 0 ${dz} units box
 run 0
 variable tmp equal pe
 variable tmp2 equal c_3
 if "(${tmp} <= ${minPE}) && (${tmp2} > 1)" then &
   "variable miniz equal ${iz}" &
   "variable minPE equal ${tmp}"
 next a
jump in.shift loopa
```

7.2.3 Flexible Molecule Quenching

The flexible stacking fault portion of the simulation further reduces the energy of the rigid stacking fault structure by replacing layers around the interface with fully flexible molecules. The fully flexible molecules undergo deformations that reduce the stacking fault energy.

The computes used to calculate the minimum pair distances for the rigid stacking faults are no longer needed for this portion of the simulation and are therefore turned off using the LAMMPS

command "uncompute". These computes are expensive to conduct and turned off to speed up the simulation. The "thermo_style" is also redefined to no longer output data from the computes, "c_2" and "c_3". A new variable "s" is defined to output dump data only using a log frequency. This is useful because initially the structure changes very quickly as the molecules on the interface are changed from rigid to flexible. After a few 100 timesteps, the minimum energy configuration is reached and the molecules stop moving. Two "dump" commands are used to create different amounts of atomic configuration data. The "dump 1" outputs the entire structure at the end of the simulation, which contains mostly frozen molecules. The "dump 2" outputs only the layers surrounding the stacking fault at log frequency intervals given by the variable "s". This is useful for watching the molecules during the quenching process. This completes all of the changes to the commands used to output simulation data.

The rigid stacking fault structure is then separated to the minimum energy separation distance using "displace_atoms". At the end of the rigid stacking fault portion of the simulation the top half was displaced by 10 Å to the position given by "${iz}". The minimum energy seperation is then found by moving the top half from its current location at "${iz}" to "${miniz}" calculated by variable "${tmp}".

New groups, "tmp_top" and "tmp_bot", are then defined that omit the mobile regions from the top and bottom halves. This is used to separate the structure into the rigid lattice and flexible molecule groups that will be treated differently during the molecular dynamics integration. The groups "tmp_top" and "tmp_bot" are to be treated as rigid molecules, and therefore to speed up the computation, the bonds and pair interactions can be removed using the following commands:

```
delete_bonds tmp_top multi

neigh_modify exclude group tmp_top tmp_top
```

Similar commands are used for "tmp_bot". The stress that is read into the variable "s_zz" from the command line is then converted to a force per atom and applied to all of the atoms in the "tmp_top" rigid lattice using the LAMMPS command "fix addforce".

Next, values are given that apply to the molecular dynamics integration of the flexible stacking faults. First, the velocity of all atoms is set to zero. All velocities should already be zero and this is just a check. Next, a 10-ps timestep is input using the "timestep 10" command. Using normal velocity Verlet molecular dynamics, this timestep is too big and there is no way the simulation would run. This is almost the same as the period of the C-H bonds, and for energy to be adequately conserved, the 10 timesteps should be taken per period of the highest vibration interaction making the velocity Verlet timestep on the order of 1 fs. In this work, a reversible reference system propagator algorithms (RESPA) multi-timestep algorithm is used to split up the timestep calculations up into three separate parts using the following command:

```
run_style respa 3 2 4 bond 1 pair 2 kspace 3
```

Using this command, only the kspace interactions are updated every 10 timesteps, the pairwise terms are updated 4 times for every update of the kspace terms and the bond terms are updated 2 times for every update of the pairwise terms or 8 times for every update of the kspace terms. Overall, the timestep for integrating the bonded terms is 1.25 ps, the pairwise terms is 2.5 ps, and the Ewald/kspace terms is 10 ps. This large timestep in the kspace term would not lead to good energy conservation of the system and poor statistical molecular dynamic trajectory data. However, this work is concerned with quenching stacking fault structures and this is an adequate method of speeding up the simulations. The RESPA style increases the overall amount time spent communicating data between compute nodes but reduces the amount of time spent calculating the Ewald sum. One problem with this method may occur due to the pair lists only being updated on the full 10-fs timesteps, but this was not observed in the work by Munday (*4*, *6*).

This work was greatly simplified using the rigid body integrators available in LAMMPS, which allowed the top rigid slab "`tmp_top`" shown by the red layer labeled "rigid" in figure 9b to be integrated along with the flexible molecules. Allowing the slab to move in the z-direction reduces unwanted axial strains and stresses on the stacking fault interface that would have resulted if the top slab had been frozen at the separation distance of Δ_r. The "`tmp_top`" atoms are integrated using the "`fix rigid`" rigid body integrator with the keyword "single", indicating the entire group of atoms this fix is applied are treated as a single rigid body. The flags following the "`force`" and "`torque`" keywords are used to turn off or on certain components of the forces. All torques and forces are turned "`off`" except for the force zflag = "`on`". This causes "tmp_top" to only be acted upon by a force in the z-direction and therefore will only move in the z-direction, i.e., "`tmp_top`" acts as a piston on the flexible stacking fault atoms used to maintain the crystal structure of the perfect crystal, enforce the stacking fault, and maintain the proper pressure. The "`1`" indicates which rigid body to apply the force constrains to. In this case, there is only a single rigid body and therefore all constrains are applied to the "`1`" rigid body.

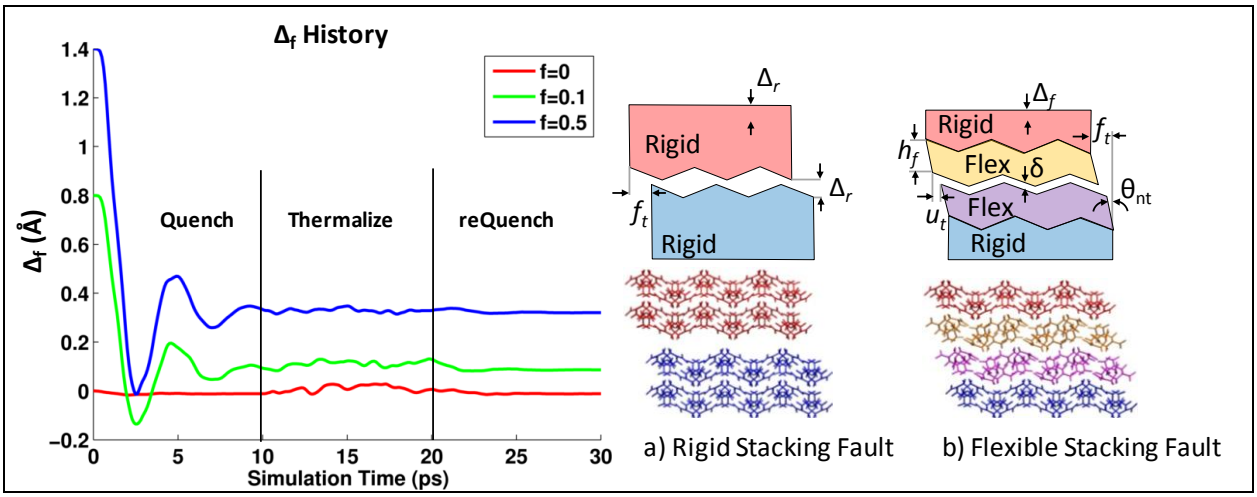

Figure 9. Axial displacement of the top rigid slab during the rigid to flexible stacking fault quenching process. Each line represents a separate (010)[100] stacking fault simulation given by the stacking fault vector in the legend. (a) Rigid stacking fault and (b) flexible stacking fault after quenching.

In figure 9, at t = 0 ps, the interface is separated to the Δ_r, as shown in (a), which gives the minimum energy configuration in figure 8. For 0 ps < t < 10 ps, layers 6 and 7 are converted to flexible molecules, as shown in (b), and the structure is quenched. This causes the interface to quickly close as the molecules shear and change conformation to minimize repulsive interactions. The quenching process is done using viscous damping to remove kinetic energy. By t = 10 ps, the flexible molecules have settled to their minimum energy configuration. For 10 ps < t < 20 ps, the flexible molecules layers are thermalized to T = 10 K to help to provide the molecules with additional kinetic energy to escape any local minima they may have become trapped in during quenching. For 10 ps < t < 20 ps, the flexible molecules layers are requenched to the final flexible stacking fault structure.

The atoms in the bottom rigid slab "`tmp_bot`" shown by the blue region labeled "rigid" in figure 9b are not included in the integration of the equations of motion and therefore will not move during the quenching process. This treats the "`tmp_bot`" atoms as a frozen body.

The flexible molecules in the "`mobile`" group are integrated using an "`nve`" ensemble. In this case, the crystal dimensions in the xy-plane is fixed while the z-dimension is allowed to change by treating "`tmp_top`" as a rigid body that moves in the z-direction. Viscous damping is then applied using "`fix viscous`" to the motion of every atom in the simulation cell to slowly freeze out the motion of the system providing a crude energy minimization. A viscous damping coefficient of "`20.0`" or ~80e8 N·s/mol-m is used to slowly damp out the motion shown by the reduction in oscillations of the Δ_f in figure 9. This damping coefficient is sufficient to damp out the motion during the "`run 1000`" steps (10 ps) of the quenching simulation. A larger damping coefficient would reduce the simulation time but may also effect the relaxation of the flexible molecules causing them to get stuck in a local minimum. A very large damping coefficient will overdamp the system resulting in a longer simulation time and an unrealistic relaxation. A

smaller damping coefficient will also require a longer simulation time to remove the atomic motion but may also allow the atoms more freedom to find the minimum energy configuration. In this work, very little difference was found in the minimum potential using smaller damping coefficients and so "`20.0`" was used.

A rethermalizaiton of the quenched structure was then performed to help the quenched structure escape any local minimums in order to settle into a configuration closer to the global minimum. The increase in kinetic energy results in a rise in the potential energy of the system, reflected by the Ψ and Φ energy shown in figure 10a. The kinetic energy does not affect the shear strain energy (*E*) in figure 10a or the stresses in figure 10b. The viscous damping is removed during thermalization of the quenched structure using "`unfix 3`". The velocity for the mobile atoms is then sent to the equivalent of T=20 K using "`velocity mobile create 20.0 2349851`", where "`2349851`" is a seed number for the random number generator used to create the velocity distribution on the mobile atoms. The Langevin thermostat is used to maintain the temperature of the mobile atoms at T = 20 K during the 1000 timesteps (10 ps) of the simulation. The random forces applied by the Langevin thermostat may help some configurations to escape local minimums better than a standard NVT ensemble. Using the Langevin thermostat for the mobile atoms also allows the entire system to be integrated by an NVE ensemble. The "`tmp_top`" group is still treated as a single rigid body that only moves in the z-direction. The increase in temperature leads to a small amount of thermal expansion of the flexible layers reflected by the slight increase in value of Δ_f for 10 ps < t < 20 ps in figure 9, where the noise is caused by thermal oscillations.

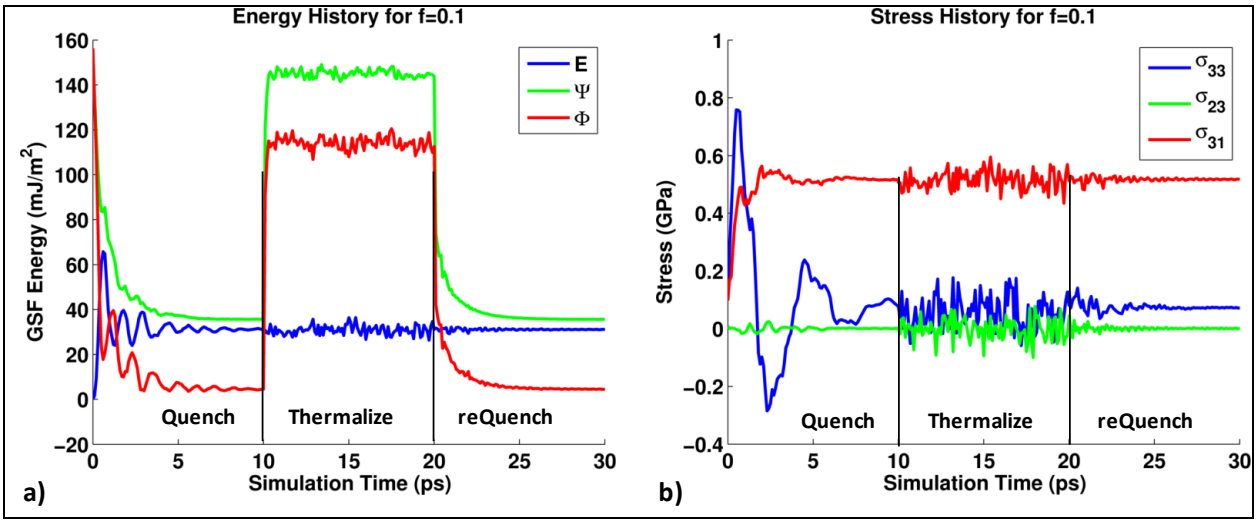

Figure 10. Energy and stress time history of the quenching process for the (010)[001] stacking fault with $f_{100} = 0.1$ shown in figure 9 by the green line. (a) Stacking fault energy components, where Ψ is the energy given by equation 1.1, *E* is the elastic strain energy of the flexible layers given by equation 1.2, and Φ is the interfacial stacking fault energy given by Φ = Ψ–*E* presented in equation 1.3. (b) Stress components of the flexible molecule layers, where the unit basis are given by e_1 = [100]/|a|, e_2 = [001]/|c|, and e_3 = [010]/|b|.

In figure 10, the flexible layers in the $f_{100} = 0.1$ stacking faults undergo significant shear allowing the interface to completely unslip back to the origin. This results in most of the stacking fault energy due to the mismatched interface being converted to elastic strain energy, i.e., at t = 10 ps & 30 ps in (a) $E \approx \Psi$ and $\Phi \approx 0$, and in (b) $\sigma_{31} \neq 0$. It is also shown in (a) by comparing energy values for t=10 ps and 30 ps that the thermalization and requenching process do little to change the final energy of the system and are unnecessary steps.

The rethermalized structure is then quenched again to find what is hoped to be a lower minimum energy configuration. The Langevin thermostat is turned off using "`unfix 3`", the velocity of the mobile atoms is set to 0, a viscous damping of 20 is then reapplied and the system is quenched for 10 ps. The requenching process occurs quickly because the configuration of this structure is closer to the minimum energy configuration than that of the original rigid stacking fault. As previously mentioned, the energy in figure 10a at t = 10 ps is almost identical to the energy at t = 30 ps indicating that the rethermalization does not improve the minimization process. The input file is shown below, where # and red letters indicate comments and are skipped over by LAMMPS.

```
#----- DISPLACE TO MINIMUM ENERGY RIGID SEPARATION AND QUENCH -----
uncompute 2
uncompute 3
thermo_style custom step dt temp c_mobtemp &
             vol lx ly lz xy xz yz &
       press pxx pyy pzz pxy pxz pyz pe ke enthalpy &
             evdwl ecoul epair ebond eangle edihed eimp emol elong &
             c_1[4][1] c_1[4][2] c_1[4][3] &
             c_1[5][1] c_1[5][2] c_1[5][3] &
             c_1[6][1] c_1[6][2] c_1[6][3] &
             c_1[7][1] c_1[7][2] c_1[7][3] &
             c_1[8][1] c_1[8][2] c_1[8][3] &
             c_1[9][1] c_1[9][2] c_1[9][3]
thermo_modify     norm no


variable    s equal logfreq(10,3,10) #dump variable

dump        1 all custom 3000 all.${ix}_${iy}.lammpstrj id type xu yu zu
dump        2 dout custom 3000 mid.${ix}_${iy}.lammpstrj id type xu yu zu
dump_modify 2 every v_s first yes #dump on 0,10,20,30,100,200,300,1000,etc

### SEPARATE TO RIGID STACKING FAULT
variable tmp equal ${miniz}-${iz}
displace_atoms top move 0 0 ${tmp} units box

### get rid of unused bonds & pairs
group tmp_top  subtract top mobile  #mobile rigid group on top
group tmp_bot  subtract bot mobile  #rigid group on bottom for pair delete
delete_bonds  tmp_top multi
delete_bonds  tmp_bot multi
neigh_modify  exclude group tmp_top tmp_top
neigh_modify  exclude group tmp_bot tmp_bot
```

```
### RUN STYLE & TIMESTEP

#Add force distributed over frozen group tmp_top
#convert force from GPa to unit style real force units (kcal/mol-A)
#3780 is number of atoms in group tmp_top
variable tmp equal -${szz}*0.1439*${lx0}*${ly0}/3780
fix 4 tmp_top addforce 0 0 ${tmp}

### RUN INITIAL QUENCH
velocity all create 0.0 2349851
timestep 10.0
run_style respa 3 2 4 bond 1 pair 2 kspace 3
fix 1 tmp_top rigid single force 1 off off on torque 1 off off off
fix 2 mobile nve
fix 3 all viscous 20.0
run 1000

### THERMALIZE QUENCHED STRUCTURE
unfix 3
velocity mobile create 20.0 2349851
fix 3 mobile langevin 20.0 20.0 100.0 699483
run 1000

### REQUENCH THERMALIZED STRUCTURE
unfix 3
velocity mobile create 0 2349851
fix 3 all viscous 20.0
run 1000
```

7.3 Generalized Stacking Fault Scripts

The LAMMPS input file presented in section 6 is for a single stacking fault vector. The (x,y) offset of the stacking fault is determined by the variables (ix,iy) passed into LAMMPS as variables using the command line option –v. Using the command line option, the same LAMMPS input file used to run a single stacking fault is run for a series of different (ix,iy) values that span an entire unit cell on the stacking fault surface. This produces the entire GSF energy surface. The dimensions of the unit cell to be spanned by stacking faults are determined within the LAMMPS input script and (ix,iy) refer to normalized values of $0 \leq ix \leq 100$ and $0 \leq iy \leq 100$. The scripts presented in this section implement this idea by running a series of simulations using a loop over a grid of ix and iy values from $0 \leq ix \leq 100$ and $0 \leq iy \leq 100$. For the (010) surface, limiting iy = 0 and $0 \leq ix \leq 100$ produces a trace of the GSF surface, as shown in figure 11, given by the rigid stacking fault simulation described in section 7.2.2 and the flexible stacking fault described in section 7.2.3. The black data points refer to the minimum energy configuration on the energy separation curve given in figure 8 and the red data corresponds to the final configuration given at either t = 10 ps or 30 ps in figure 10a. Methods for post-processing the flexible data Ψ_F to remove the effects of shear using the equations presented by Rice (*8*) and is presented in section 7.4.3.

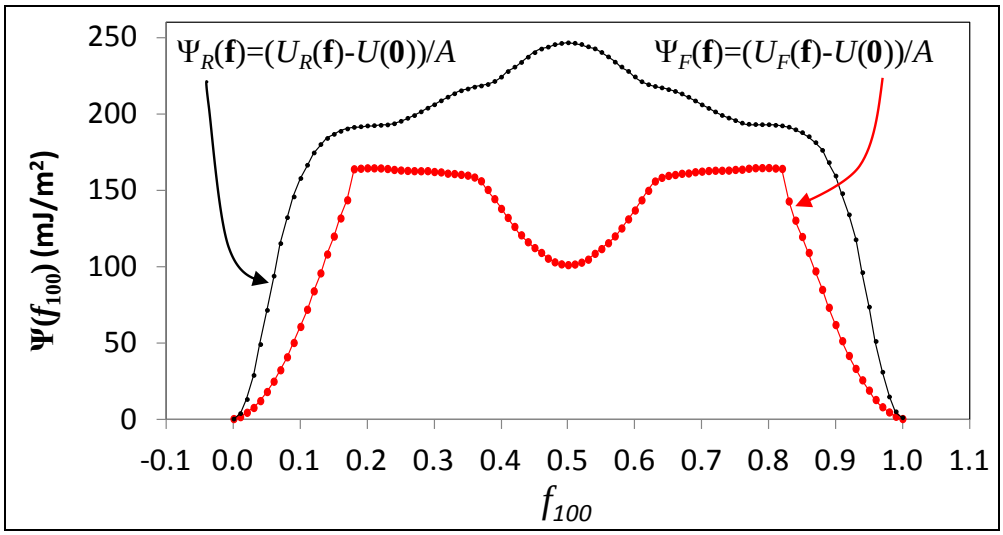

Figure 11. (010)[100] b2 GSF energy for the rigid lattice in black, Ψ_R, and flexible lattice in red, Ψ_F. U is the potential energy and A is the area of the stacking fault interface. The black line, Ψ_R, is given by the minimum energy Δ_r given by the energy separation curves in figure 8 for a series of stacking faults spanning one unit cell in the [100] direction. The red line, Ψ_F, is the final quenched configuration given for t = 10 ps or 30 ps by the green line in figure 10a.

7.3.1 CSH Script for Running Arrays of SF Simulations

FILE: run.GSF

A CSH script is presented here that runs several stacking fault simulations in series on a local multicore Linux computer like the Dell computer described in the makefile given in section 4. This CSH script is used to quickly test the pbs script presented in section 7.3.2 and therefore accomplishes the same tasks. This script copies the LAMMPS input files to the correct directory to be executed by LAMMPS as was done by the CSH script in section 6.10. In this script, arrays are created, "`$ARRAYx`" and "`$ARRAYy`", containing the (`ix,iy`) coordinates of the stacking fault simulations. The arrays are used to create a grid of stacking faults and every `ix` value is run for every `iy` value creating an `ix`×`iy` grid of stacking faults. The values `ix` and `iy` are passed into LAMMPS as variables using the `-v` LAMMPS command line directive used when executing the simulation. The variables (`ix,iy`) are the normalized stacking fault vectors divided by 100, i.e., `ix` = 023 is equal to a stacking fault vector $f_x = 0.23$ and likewise for `iy` and f_y. In section 7.2.1, (`ix,iy`) are converted to the real space offset of the stacking fault vector (`dx,dy`). The strings given in "`$ARRAYx`" and "`$ARRAYy`" are also used to redirect the LAMMPS thermo output files to the file named `log.ix_iy` using the LAMMPS command line directive "`-log log.${ix}_${iy}`". Redirecting the log files allows every simulation to write data to the same directory without overwriting previous stacking fault simulation data files. The dumpfiles use a similar naming convention. The naming convention used in `$ARRAYx` and `$ARRAYy` must strictly follow a "`%3s`" format for the values.

Creating the (ix,iy) grid is most easily accomplished using two nested "LOOPS" but to match the limitations of pbs, only a single loop is used here. To mimic the effect of nested loops, the variable "c" is used to parse the correct values out of the arrays for ix and iy. These variables along with szz are then passed to the LAMMPS simulation during execution. This script runs in serial so a new stacking fault simulation is not run until the previous stacking fault simulation is complete. Each simulation takes about 30 min to run on eight processors. Running a 11×11 grid takes about 3 days. In the CSH script shown below, the # and blue letters indicate comments and are skipped over.

```
#!/bin/csh

###USER INPUT
set szz=2.50
set IN=in.shift

###SET READ WRITE DIRECTORIES
set HOMEDIR=/data/lammps-27Oct11/data/RDX/Shear_therm
set DATADIR=${HOMEDIR}/RESULTS.b_zdir
set RESDIR=${DATADIR}/RESULTS.junk

### GET RESDIR READY FOR RUN
mkdir ${RESDIR}
cp ${HOMEDIR}/${IN} /${RESDIR}/${IN}
cp ${DATADIR}/data.b2_nz6 /${RESDIR}/data.nz6
cp ${HOMEDIR}/../POTENTIALS/potential_ewaldn.mod /${RESDIR}

echo RESDIR=${RESDIR}
cd ${RESDIR}

### SET EXECUTABLE PATH & LOAD MODULES FOR MPI
module load cse/openmpi/1.4.1
set EXE=/data/lammps-27Oct11/src/lmp_LynnDell

### SET UP STACKING FAULT ARRAYS AND RUN ALL OF THEM
set ASIZEx=4          #size of ix array
set ARRAYx=( 010 020 030 040 ) #ix
set ARRAYy=( 000 )       #iy
set RUNTOT=4          #total runs=ix*iy

set c=1
while ( $c <= ${RUNTOT} )
 @ i = ${c} - ${ASIZEx} * ( ( ( ${c} - 1 ) / ${ASIZEx} ) )
 @ j = ( ( ${c} - 1 ) / ${ASIZEx} ) + 1
 set ix=$ARRAYx[${i}]
 set iy=$ARRAYy[${j}]

 echo "%%%"
 echo "%%%"
 echo "******** RUNNING ix= ${ix} iy= ${iy} szz= ${szz} **************"
 echo "%%%"
 echo "%%%"
 mpirun -np 8 ${EXE} -log log.${ix}_${iy} -v szz ${szz} -v ix ${ix} -v iy
${iy} < ${IN}
```

```
 @ c = $c + 1
end
```

7.3.2 PBS Script for Submitting Arrays of SF Simulations

FILE: qsub.GSF

This script is used to submit an array of simulations to the batch queuing system, pbs, on the ARL cluster Harold. This pbs script performs a similar function on the ARL clusters as the CSH script in section 7.3.1 does on the local Linux desktop computer. The top lines in the script containing "`#PBS -…`" are not commented out but are pbs commands. The other top lines in blue with "`#`" followed by a space are comments and are skipped over by pbs. More information on the pbs commands used below is available at the ARL DSRC Web site: http://www.arl.hpc.mil/docs/pbsUserGuide.html. The ARL Web site also gives several pbs commands used for killing and holding batch jobs, checking the status of jobs, and the number of jobs waiting on the queue. This section only presents commands that are different from the pbs script used to submit a single LAMMPS simulation presented in section 6.8.

This pbs script uses the pbs array option, the main difference between this pbs script and the one presented in section 6.9. This allows an array of simulations to be submitted using a single pbs script. The pbs arrays used in conjunction with the LAMMPS command-line variables provides an efficient method of submitting hundreds of different stacking fault configurations into the batch queuing system. Programming "LOOPS" cannot be used in pbs and in their place the pbs variable "`${PBS_ARRAY_INDEX}`" is used. The "`${PBS_ARRAY_INDEX}`" index is used to loop over all of the SF configurations, (`ix,iy`), to be run. Values for `ix` are hard coded into the arrays `ARRAYx` and likewise for `iy` and `ARRAYy`. It is assumed that a grid of stacking faults is being created and therefore every `iy` is run for every `ix`. The "`$ARRAYx`" and "`$ARRAYy`" shown in the following script will create an 11x11 grid of stacking faults with stacking faults created with 0.1 grid spacings between stacking faults. The stacking fault vector (`ix,iy`) input into LAMMPS is normalized and divided by 100, i.e., `ix` = 023 is equal to a stacking fault vector f_x = 0.23 and likewise for iy and f_y. The "`$ARRAYx`" and "`$ARRAYy`" is also used to name the LAMMPS output file to be created for each stacking fault and the naming convention must strictly follow a "`%3s`" format for the values entered into the arrays.

Creating the (`ix,iy`) grid is most easily accomplished using two nested "LOOPS", but the pbs array is only able to mimic a single loop. To get around this, the variable "`c`" is used to parse the correct values out of the arrays for `ix` and `iy`. The "`${PBS_ARRAY_INDEX}`" is then used to create a series of (`ix,iy`) stacking faults and every stacking fault structure is then submitted to the queue. The runtime on these simulations is short and there is usually a very little wait time in the queues. It normally takes less than 3 h for all 121 simulations in the following pbs script to

run. In the pbs script shown below, the # and blue letters indicate comments and are skipped over.

```
#!/bin/csh
# Request maximum wallclock time for the job
#PBS -l walltime=01:00:00
# select=number_of_nodes,ncpus=cores/node,mpiprocs=MPI_procs/node
# Total cores requested = number_of_nodes X MPI_procs/node
#PBS -l select=1:ncpus=8:mpiprocs=8
# Request job name: 010=SF plane; b2=SF interface; 2.50=szz axial load GPa
#PBS -N almp010b2P2.50
# Array job. Number of stacking fault simulations to run
#PBS -J 1-121:1
# Request the PBS job queue for the job
#PBS -q standard
# Specify how to distribute MPI processes across nodes
#PBS -l place=scatter:excl
# Combine the output and error files into a single file
#PBS -j oe
# Select Project ID
#PBS -A PROJECTID###
# Request environment variables be exported from the script
#PBS -V
#
#
############################
# script for LAMMPS
############################
#
# Array jobs are identified by PBS_JOBNAME and PBS_ARRAY_INDEX
echo "running job index ${PBS_ARRAY_INDEX}"
echo "PBS_JOBNAME=${PBS_JOBNAME}"
#
# Setting JOBID to use job number PBS has JOBID[#].o2
set JOBID=`echo "$PBS_JOBID" | cut -f1 -d[`
set DIR=`echo "${PBS_JOBNAME}" | cut -c5-7`
set BASIS=`echo "${PBS_JOBNAME}" | cut -c8-9`
set szz=`echo "${PBS_JOBNAME}" | cut -c11-14`
echo "started job ${JOBID}.${PBS_ARRAY_INDEX} on" `date`
echo "GSF plane normal: DIR=${DIR}"
echo "GSF plane BASIS=${BASIS}"
echo "Uniaxial stress szz=${szz}"
#
# MANUALLY INPUT THE STACKING FAULT ARRAY VALUES FOR (ix,iy)
# The arrays values are used also used to name the files
# and a %3s format must be used
# size(ix)*size(iy) equals the number #PBS -J 1-121:1
# This assumes that all iy values will be run for each ix value (grid)
##--STACKING FAULT ARRAY------
set ASIZE=11 #size of ARRAYx
set ARRAYx=( 000 010 020 030 040 050 060 070 080 090 100 )
set ARRAYy=( 000 010 020 030 040 050 060 070 080 090 100 )
# in place of nested loop over grid of ARRAYx by ARRAYy
set c=${PBS_ARRAY_INDEX}
      @ i = ${c} - ${ASIZE} * ( ( ( ${c} - 1 ) / ${ASIZE} ) )
```

```
      @ j = ( ( ${c} - 1 ) / ${ASIZE} ) + 1
      set ix=$ARRAYx[${i}]
      set iy=$ARRAYy[${j}]
echo STACKING FAULT: ${ix}_${iy}
#
# Set name of LAMMPS input file
set IN=in.shift
echo "lammps input file name IN=${IN}"
#
# Set file paths for input and output
# in RESDIR, 1L refers to the number of flexible SF unit cells layers
# Only a single results directory is created and will contain all of
# the ix and iy data
set HOMEDIR=/usr/people/lmunday/Lammps/lammps-21Dec11/data/RDX
set DATADIR=${HOMEDIR}/bGSFsurf/RESULTS.${DIR}
set RESDIR=${DATADIR}/RESULTS.yphase1L${BASIS}_szz${szz}
echo HOMEDIR is ${HOMEDIR}
echo RESDIR is ${DATADIR}
echo RESDIR is ${RESDIR}
#
# Temporary directory where simulation is run
# each qsub array index run in a separate directory
set TMPD=/usr/var/tmp/lmunday/${JOBID}.${PBS_ARRAY_INDEX}
mkdir -p ${TMPD}
echo TMPD is ${TMPD}
#
# Copy LAMMPS data over to tmp directory
cp ${DATADIR}/../${IN} /${TMPD}/${IN}
cp ${HOMEDIR}/zPOTENTIALS/potential_ewaldn.mod /${TMPD}
cp
${HOMEDIR}/aMinimize/RESULTS.${DIR}/RESULTS.yphase${BASIS}szz${szz}/data.matl
ab /${TMPD}/data.nz6
#
cd ${TMPD}
echo
echo contents of ${TMPD} before start is:
ls -l
echo
#
# LOAD MODULES FOR MPI
#
unlimit
module load compiler/intel11.1
module load mpi/sgi_mpi-1.26
# Execute simulation.
#
# Using LAMMPS command line directive -log to name log file by the stacking
# fault vector, ix_iy. Every stacking fault log file is saved to the same
# folder and this keeps the separate stacking faults log files from
# overwriting one another. The dump file also uses the ix_iy format.
# Using the LAMMPS command line directive -v to pass in three variables,
# stacking fault vector, (ix,iy) and the stress normal to the SF plane (szz)
echo starting program execution on `date`
set EXE=${HOMEDIR}/../../src/lmp_harold
mpiexec_mpt ${EXE} -log log.${ix}_${iy} -v ix ${ix} -v iy ${iy} -v szz ${szz}
< ${IN}
echo completed program execution on `date`
```

```
echo
echo contents of ${TMPD} after completion is:
ls -l
echo
#
# Moving the entire result directory back. This will overwrite previous
# stacking fault LAMMPS input files but will not overwrite previous run log
# and dump files because they are named differently i.e. log_ix_iy.
mkdir ${RESDIR}
mv * ${RESDIR}/
cd ..
rm -r ${TMPD}
#
cd ${RESDIR}
echo contents of results directory is:
ls
echo
echo completed job ${JOBID}.${PBS_ARRAY_INDEX} on `date`
```

7.4 Matlab Scripts for Post-Processing Stacking Fault Data

This section presents the Matlab scripts used to parse the LAMMPS log files for all of the stacking fault simulations, post-process it to determine the effects of the flexible layers, and produce plots of the data. Figure 11 presented the stacking fault data as calculated by the equations provided by Vitek (*7*). Figure 12 shows the amount of shear stress and strain that results from the introduction of a flexible layer of molecules into the simulation around the stacking fault. The strain calculation is determined from the center of mass displacement of the layer data. Rice (*8*) presents a modified calculation to account for the effects of shear in the GSF energy by determining the interfacial displacement and interfacial stacking fault energy as presented in figure 13. Figure 14 shows a comparison of the initial rigid stacking fault energy and the final flexible stacking fault energy. The lines connecting the two data sets are the relaxation histories for single stacking faults during the quenching process.

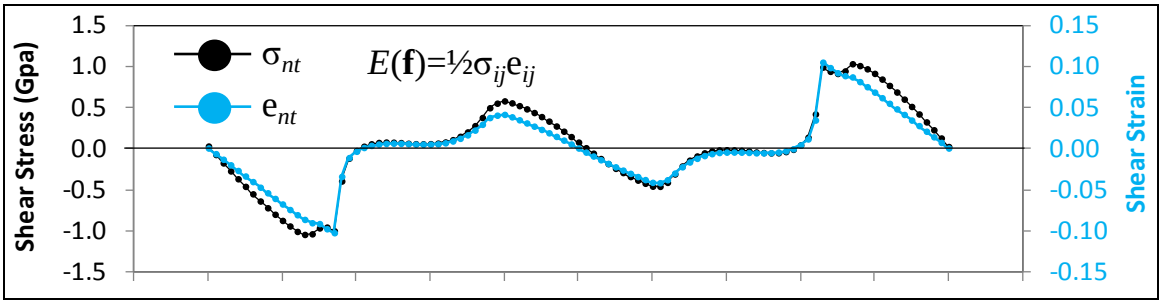

Figure 12. Shear stress and strain components for the (010)[100] b2 flexible GSF data given by the red line in figure 11. These are the final stress and strain values given at the end of the quenching process *i.e.* $\sigma_{nt} = \sigma_{31}(t = 10\ ps)$ in figure 10a. The elastic strain energy, *E*, given by the product of stress and strain, is shown by the blue line in figure 10a.

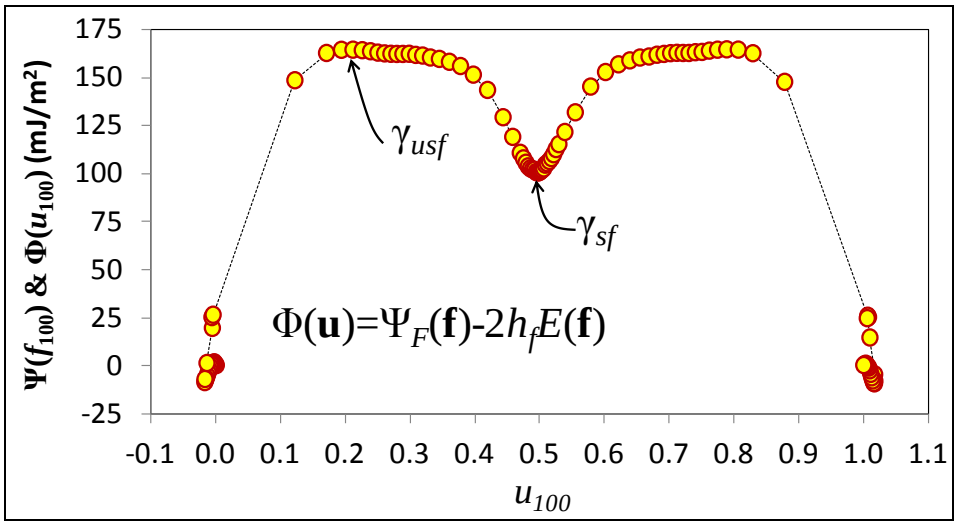

Figure 13. (010)[100] b2 Interfacial GSF energy, $\Phi(u_{100})$, plotted as a function of the interfacial displacement, u_{100}. The unstable stacking fault energy is indicated by γ_{usf} and the stable stacking fault energy is indicated by γ_{sf}.

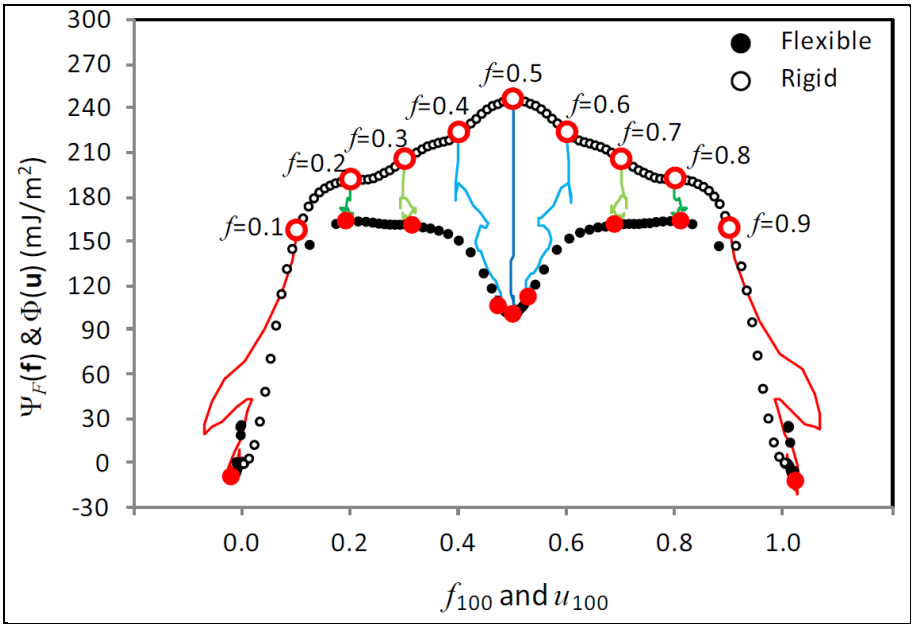

Figure 14. Rigid $\Psi_R(f_{100})$ and flexible $\Phi(u_{100})$ energy surfaces where lines indicate the relaxation path of the rigid to flexible stacking fault. Open symbols denote the initial rigid configurations and the closed symbols indicate the subsequently relaxed flexible configurations. Relaxation paths for select rigid to flexible stacking fault structures are shown by the lines connecting the larger symbols given at f_{100} to their final relaxation position at u100. Relaxation histories are given in 0.1 increments of f_{100}. Time history data for f = 0.1 is given in figure 10.

7.4.1 Matlab Script for Reading Log Files for Series of Stacking Faults

FILE: read_log_xy.m

The purpose of this Matlab script is to read in data from every file in a directory with a log* name. Several arrays of data from the thermo output are created that are used by subsequent Matlab scripts to determine the GSF energy curves. The `B` arrays contain data from the energy separation curves shown in figure 8 and are created from the portion of the LAMMPS input script given in section 7.2.2. The `C` arrays contain time history data for the first rigid to flexible stacking fault shown in figures 9 and 10 and are created from the portion of the LAMMPS input script given in section 7.2.3. The `D` and `E` arrays contain the thermalization and requenching data but are not often used. The `Z` array contains the concatenation of arrays `C`,`D`,`E`.

From the first quenching simulations the array `Cstart` contains all of the initial rigid stacking fault data for all of the stacking faults contained in the directory. The array `Cfinal` contains the final data for all of the stacking faults. The output thermo data contained in each array is printed to the Matlab command prompt when the file is run. The array `C` contains all of the history data for the entire quenching process from the rigid stacking fault given by `Cstart` to the final quenched configuration given by `Cfinal` for every stacking fault simulation. The data for each stacking fault in C are separated by a `NaN`. These arrays are used in the plot_history.m, plot_trace.m, and plot_xy.m files to post-process the data to determine and plot the GSF energy curves.

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% READ IN ENTIRE XY GSF SURFACE
% READS IN ONLY ONE LAYER THICKNESS, either 1 or 2 layers
% FOR RIGID & FLEXIBLE PORTIONS OF SIMULATION
%
%
% USED BY:
% -plot_xy.m to produce GSF contour plots
% -plot_trace.m to produce traces of the GSF contours
% -plot_history.m to produce time histories of variable during quench
%
% each log file created by in.shift contains data in this order:
% STATIC RUNS (Single line of output):
% (1)      -Initial configuration ie no seperation or stacking fault
% (2:end-3)   -Rigid opening with SF (large increments to free surface)
%
% DYNAMIC RUNS (dynamics to minimize SF at min rigid opening)
% (end-2)   -Quench stacking fault starting at minimum rigid opening PE
% (end-1)   -Thermalize quenched structure to get rid of local minimums
% (end)    -reQuench thermalized structure to find new minimum.
%
%
% Matlab functions used:
% - textscanlog.m available in Lammps folder /tools/Matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
close all hidden; clear all; clc; fclose all;
lammps_path='/home/lmunday/download/SF/RDX';
addpath([lammps_path,'/zzMatlab/LammpsMatlab']) %read lammps log
%
% MANUAL INPUT
%
%File labels
fdir='001';  % Slip Plane
flayer=1;   % Number of flexible layers (1 or 2)
fbase='b1';  % b1 or b2 slip plane
fname='szz1.00';  % Uniaxial stress applied normal to slip plane

file_path=[lammps_path,...
'/bGSFsurf/RESULTS.',fdir,'/RESULTS.',num2str(flayer),'L',fbase,'_',fname]

%#ok<*SAGROW>   %this comment suppresses increasing arrays in loops warning
%#ok<*ST2NM>   %this comment suppresses str2num warnings

%%%%--Get log files and sort files by index number
%------------------------------------------------------------------------
% Create list of logfiles to be read in
logfiles = dir(fullfile(file_path, '/log*'));
%store input file increment numbers into nx2 cell
for i=1:size(logfiles,1) %Assuming logfile name is formatted as "log.###.###
 ixy(i,1)=str2num(logfiles(i).name(5:7));
 ixy(i,2)=str2num(logfiles(i).name(9:11));
 filesize(i)=(logfiles(i).bytes); %look for small files w/ errors
end
index0=find(filesize < 1000);
if (size(index0,2) > 0), fprintf('Incomplete files: \n'), end
fprintf(' %s \n',logfiles(index0).name)
ixy(index0,:)=[];
%
%------------------------------------------------------------------------
% Read data into appropriate arrays
Atot=[];
Btot=[];
Ctot=[];
Dtot=[];
Etot=[];
Ztot=[];
for j=1:size(ixy,1)
 file_name=['log.',sprintf('%03d',ixy(j,1)),...
        '_',sprintf('%03d',ixy(j,2))]; %padding ixy with zeros
 file_in=[file_path,'/',file_name];
 logdata = textscanlog(file_in);
 %
 %%% initialization structure data -- mostly worthless
 Atot=vertcat(Atot,cell2mat(logdata.data(1,:)));
 %
 %%% Concatenate Rigid opening simulations
 B=[];
 % skip first one because it is the bulk unfaulted structure
 for i=2:size(logdata.Chead,1)-3
  B=vertcat(B,cell2mat(logdata.data(i,:)));
 end
 Btot=vertcat(Btot,B,NaN(1,size(B,2))); % put NAN to separate data
```

```
 Bstart(j,:)=B(1,:);
 Bfinal(j,:)=B(end,:);
 %
 %%% Flexible quenching simulations
 C=[];
 C=cell2mat(logdata.data(i+1,:));
 Ctot=vertcat(Ctot,C,NaN(1,size(C,2))); % put NAN to separate data
 Cstart(j,:)=C(1,:);
 Cfinal(j,:)=C(end,:);
 %%% Flexible retherm
 D=[];
 D=cell2mat(logdata.data(i+2,:));
 Dtot=vertcat(Dtot,D,NaN(1,size(D,2))); % put NAN to separate data
 Dmean(j,:)=mean(D,1);
 %%% Flexible requench
 E=[];
 E=cell2mat(logdata.data(i+3,:));
 Etot=vertcat(Etot,E,NaN(1,size(E,2))); % put NAN to separate data
 Estart(j,:)=E(1,:);
 Efinal(j,:)=E(end,:);
 %%% TOTAL [Quench,Therm,Quench]
 Z=vertcat(C,D,E);
 Ztot=vertcat(Ztot,Z,NaN(1,size(Z,2))); % put NAN to separate data
 Zstart(j,:)=Z(1,:);
 Zfinal(j,:)=Z(end,:);
end
%
%-------------------------------------------------------------------------
%
% print out recorded data
fprintf('Number of simulation fixes: %d \n',size(logdata.Chead,2))
for i=1:size(logdata.Chead,2)-1
 fprintf('%5d  %s \n',i,logdata.Chead{1,i})
end
```

7.4.2 Matlab Script for Determining GSF Energy Surface from read_log_xy.m

FILE: plot_xy.m

This Matlab file is used to post-process the stacking fault simulation data and determine the GSF energy surface from it. The center of mass data is post-processed here, as outlined in section 5.1 of Munday's Ph.D. dissertation (*4*) and in the work by Munday et al. (*6*). Only the portion of the script pertaining to calculating the flexible layer shear strain, shear strain energy *E*, interfacial displacement u, and interracial stacking fault energy Φ(u) is presented. The equation numbers presented in the script correspond to the equations in Munday's Ph.D. dissertation (*4*). Comments after variables refer to figure 2a and b in Munday's Ph.D. dissertation (*4*) as well. The entire plot_xy.m script will create 2-D contour and scatter plots of the stacking fault data. Small edits in the script are made to allow it to calculate the shear and its associated properties for stacking faults created from 1 or 2 unit cell thick layers of flexible molecules. The data presented by Munday (*4*, *6*) used only 1 layer of flexible molecules.

The Matlab script uses the quenched flexible stacking fault from the crystal created with no stacking fault, $f_x = f_y = 0$. These data are used as the reference structure and all stress, strain, and energy measurements are made with respect to this configuration. It is assumed that the first configuration contained in the array is for the perfect crystal without a stacking fault. This is usually the case, but if there is not a log.000_000 file, the structure will be made in reference to a non-perfect configuration and this is wrong. The Matlab script then uses a loop to read in each of the quenched flexible stacking fault structures and computes the strains and stacking fault energies for the stacking fault simulations. In the Matlab file shown below, the % and green letters indicate comments and are skipped over by Matlab.

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------
%
% STACKING FAULT ENERGY CALCULATION
%
% The calculations below reference equations and figures given in Section 5:
% LB Munday, University of Maryland PhD Dissertation
%   http://hdl.handle.net/1903/12254
% Results from these simulations were also presented in:
% LB Munday, SD Solares, PW Chung, Phil. Mag., accepted 2012
%-------------------------------------------------------------------------
%
%-------------------------------------------------------------------------
%
% INITIAL DEFORMED PERFECT CRYSTAL variable labels refer to Figure 5.2a
%
%-------------------------------------------------------------------------
%
%z-dimension thickness of volume containing flexible molecules
% hf in equation (5.2)
flex_h=L*(Fstart(1,39)-Fstart(1,36));          %%%%SINGLE LAYER EDIT
%fraction of simulation volume containing flexible molecules
% Vflex in equation (5.4)
vscale=Fstart(1,8)/(2*flex_h);
%initial lattice constants of compressed/strained crystal
lat(1)=Fstart(1,6)/nlat(1);
lat(2)=Fstart(1,7)/nlat(2);
lat(3)=2*(Fstart(1,39)-Fstart(1,36));

% initial vectors between COM's on bottom surface
ro_5=Fstart(1,34:36)';  %r_o^Rbot
ro_6=Fstart(1,37:39)';  %r_o^Fbot
ro_56=(ro_6-ro_5)/(2/L); %r_o^bot            %%%%SINGLE LAYER EDIT
ho_56=[1 0 ro_56(1); 0 1 ro_56(2); 0 0 ro_56(3)]; %eqn 5.8
inv_56=inv(ho_56);
so_bot=[ro_56(1)/L; ro_56(2)/L; ro_56(3)/L]; %s_o^bot  %%%%SINGLE LAYER EDIT

% initial vectors between COM's on top surface
ro_7=Fstart(1,40:42)';  %r_o^Ftop
ro_8=Fstart(1,43:45)';  %r_o^Rtop
ro_78=(ro_8-ro_7)/(2/L); %r_o^top            %%%%SINGLE LAYER EDIT
```

```
ho_78=[1 0 ro_78(1); 0 1 ro_78(2); 0 0 ro_78(3)];
inv_78=inv(ho_78);
so_top=[-ro_78(1)/L; -ro_78(2)/L; -ro_78(3)/L];%s_o^top %%%%SINGLE LAYER EDIT

%Symmetric stress tensor of deformed perfect crystal
sigo=-[Ffinal(1,13) Ffinal(1,16) Ffinal(1,17);...
    Ffinal(1,16) Ffinal(1,14) Ffinal(1,18);...
    Ffinal(1,17) Ffinal(1,18) Ffinal(1,15)];
%
%-------------------------------------------------------------------------
%
% CRYSTAL WITH STACKING FAULT
%
%-------------------------------------------------------------------------
%
clear offset fvec fvec_n uvec uvec_n e_surf
for i=1:size(Fstart,1)
 fvec(i,1)=ixy(i,1)*lat(1)/100; % Rigid Stacking Fault Vector
 fvec(i,2)=ixy(i,2)*lat(2)/100;
 fvec(i,3)=((Fstart(i,42)'+so_top(3))-(Fstart(i,39)'+so_bot(3)))'; %eqn 5.6
 fvec_n(i,:)=fvec(i,:)./lat(:)';

 offset=((Fstart(i,40:42)'+so_top)-(Fstart(i,37:39)'+so_bot))'-fvec(i,:);

 %Shear of bottom surface (between layers 5 & 6)
 r_5=Ffinal(i,34:36)'+(ro_56*(2-L));         %%%%SINGLE LAYER EDIT
 r_6=Ffinal(i,37:39)';
 r_56=r_6-r_5;
 h_56=[1 0 r_56(1); 0 1 r_56(2); 0 0 r_56(3)]; %eqn 5.9
 F_bot=h_56*inv_56;  %eqn 5.10
 s_bot=F_bot*so_bot; %projected interfacial displacement, eqn 5.11
 e_bot=0.5*(F_bot'*F_bot-eye(3)); %Lagrange strain, eqn. 5.14

 %Shear of top surface (between layers 7 & 8)
 r_7=Ffinal(i,40:42)';
 r_8=Ffinal(i,43:45)'-(ro_78*(2-L));         %%%%SINGLE LAYER EDIT
 r_78=r_8-r_7;
 h_78=[1 0 r_78(1); 0 1 r_78(2); 0 0 r_78(3)]; %eqn 5.9
 F_top=h_78*inv_78;  %eqn 5.10
 s_top=F_top*so_top; %projected interfacial displacement, eqn 5.11
 e_top=0.5*(F_top'*F_top-eye(3)); %Lagrange strain, eqn. 5.14

 % Average strain of top and bottom
 e_avg=(e_top+e_bot)/2;

 %Flexible interfacial displacement
 uvec(i,:)=((r_7+s_top)-(r_6+s_bot))'-offset(:)'; %eqn 5.12
 uvec_n(i,:)=uvec(i,:)./lat(:)';  %normalized

 %Symmetric stress tensor of quenched structure with stacking fault
 sig=-[Ffinal(i,13) Ffinal(i,16) Ffinal(i,17);...
    Ffinal(i,16) Ffinal(i,14) Ffinal(i,18);...
    Ffinal(i,17) Ffinal(i,18) Ffinal(i,15)];
 %offset stress by stress in deformed crystal w/o stacking fault
 sig=(sig-sigo);
 %Scale stress by volume fraction of flexible molecules
 sig=(sig)*vscale;
```

```
 %Convert stress from atm to GPa
 sig=(sig)*0.000101325;

 %elastic shear strain energy per unit area of stacking fault (mJ/m2)
 %--Shear strain energy only eqn. 5.3
 e_surf(i)=1/2*(e_avg(2,3)*sig(2,3)+e_avg(3,2)*sig(3,2)...
          +e_avg(1,3)*sig(1,3)+e_avg(3,1)*sig(3,1))...
        *(2*flex_h)*1e12/1e10;
 %--Shear + volumetric strain energy eqn 5.3
 e_total(i)=(sum(sum(e_avg.*sig))/2)*(2*flex_h)*1e12/1e10;
end
fvec_n=1-fvec_n;
uvec_n=1-uvec_n;
%Calculate Generalized Stacking Fault Energy, eqn. 5.1
rPE=(Fstart(:,19)-Fstart(1,19));
fPE=(Ffinal(:,19)-Ffinal(1,19));
%convert to mJ/m2
rPE=rPE*4.184*1e6/Av*1e20./(Ffinal(1,6)*Ffinal(1,7));
fPE=fPE*4.184*1e6/Av*1e20./(Ffinal(1,6)*Ffinal(1,7));
%Calculate Interfacial Generalized Stacking Fault Energy, eqn. 5.2
fPEe=fPE(:)-e_surf(:);
%
%-------------------------------------------------------------------------
%
% END OF STACKING FAULT ENERGY CALCULATION
%
%-------------------------------------------------------------------------
```

7.4.3 Matlab Script for Reading Log Files for Series of Stacking Faults

FILE: plot_trace.m

This Matlab script uses the same algorithm presented in section 7.4.2 for plot_log_xy.m. This Matlab script is able to parse out traces from the xy logdata to create figures 11–13. This script also post-processes all of the history data between `Cstart` and `Cfinal` to create the relaxation paths shown in figure 14.

7.4.4 Matlab Script for Reading Log Files for Series of Stacking Faults

FILE: plot_history.m

This Matlab script also uses the same algorithm presented in section 7.4.2 for plot_log_xy.m. This script plots the entire relaxation history stored in the `Z` array containing the initial quenching, thermalization and requenching for single stacking faults as shown in figures 9 and 10. This script also produces the energy separation curves for single stacking faults as shown in figure 8.

8. Conclusion

This manual and accompanying files provide detailed instructions for reproducing the molecular dynamics simulations performed by Munday et al. (*4*–*6*). These simulations are able to provide thermodynamic quantities such as elastic constants, coefficients of thermal expansion, and crystal structures. Procedures are also given for producing minimized crystal structures suitable for simulations requiring a crystal structure at T=0 K such as lattice dynamics and quantum simulations. In this work, the minimized crystal structures are used in GSF simulations under varying loading scenarios. This work provides the atomistic details and parameters required for mesoscale simulations of dislocations and nucleation.

9. References

1. Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computation Physics* **1995**, *117*, 1–19.
2. Choi, C. S.; Prince, E. The Crystal Structure of Cyclotrimethylene-trinitramine. *Acta Crystallographica* **1972**, *B28*, 2857–2862.
3. Smith, G. D.; Bharadwaj, R. K. Quantum Chemistry Based Force Field for Simulations of HMX. *Journal of Physical Chemistry B* **1999**, *103*, 3570–3575.
4. Munday, L. B. Computational Study of the Structure and Mechanical Properties of the Molecular Crystal RDX. *University of Maryland College Park PhD Dissertation.* [Online] 2011. http://hdl.handle.net/1903/12254.
5. Munday, L. B., et al. Simulations of High-Pressure Phases in RDX. *The Journal of Physical Chemistry B* **2011**, *115*, 4378–4386.
6. Munday, L. B.; Solares, S. D.; Chung, P. W. Generalized Stacking Fault Energy Surfaces in the Molecular Crystal aRDX. *Philosophical Magazine* **2012**.
7. Vitek, V. Intrinsic Stacking Faults in Body-centred Cubic Crystals. *Philosophic Magazine* **1968**, *18*.
8. Rice, J. R. Dislocation Nucleation from a Cracktip: An Analysis Based on the Peierls Concept. *Journal of the Mechanics and Physics of Solids* **1992**, *40*, 239–271.
9. Bedrov, D., et al. Molecular Dynamics Simulations of HMX Crystal Polymorphs Using a Flexible Molecule Force Field. *Journal of Computer-Aided Materials Design* **2001**, *8*, 77–85.
10. Sun, H. COMPASS: An ab Initio Force-Field Optimized for Condensed-Phase Applications - Overview with Details on Alkane and Benzen Compounds. *Journal of Physical Chemistry B* **1998**, *102*, 7338–7364.

List of Symbols, Abbreviations, and Acronyms

ARL	U.S. Army Research Laboratory
CSE	computational science environment
DSRC	Defense Supercomputing Resource Center
FFTs	fast Fourier transforms
GSF	Generalized Stacking Fault
HPC	high-performance computer
LAMMPS	Large-Scale Atomic/Molecular Massively Parallel Simulator
MPI	Message Passing Interface
RESPA	reversible reference system propagator algorithms
SB	Smith and Bharadwaj
VdW	van der Waals

1 DEFENSE TECH INFO CTR
(PDF) ATTN DTIC OCA (PDF)

2 US ARMY RSRCH LABORATORY
(PDFS) ATTN IMAL HRA MAIL & RECORDS MGMT
ATTN RDRL CIO LL TECHL LIB

2 US ARMY RSRCH LAB
(PDFS) ATTN RDRL CIH C
L MUNDAY
B RICE

INTENTIONALLY LEFT BLANK.